pdp11/60
processor handbook

digital

digital

DIGITAL EQUIPMENT CORPORATION, Corporate Headquarters: Maynard, Massachusetts 01754, Telephone: (617)897-5111—SALES AND SERVICE OFFICES: UNITED STATES—ALABAMA, Huntsville • ARIZONA, Phoenix and Tucson • CALIFORNIA, El Segundo, Los Angeles, Oakland, Ridgecrest, San Diego, San Francisco (Mountain View), Santa Ana, Santa Clara, Stanford, Sunnyvale and Woodland Hills • COLORADO, Englewood • CONNECTICUT, Fairfield and Meriden • DISTRICT OF COLUMBIA, Washington (Lanham, MD) • FLORIDA, Ft. Lauderdale and Orlando • GEORGIA, Atlanta • HAWAII, Honolulu • ILLINOIS, Chicago (Rolling Meadows) • INDIANA, Indianapolis • IOWA, Bettendorf • KENTUCKY, Louisville • LOUISIANA, New Orleans (Metairie) • MARYLAND, Odenton • MASSACHUSETTS, Marlborough, Waltham and Westfield • MICHIGAN, Detroit (Farmington Hills) • MINNESOTA, Minneapolis • MISSOURI, Kansas City (Independence) and St. Louis • NEW HAMPSHIRE, Manchester • NEW JERSEY, Cherry Hill, Fairfield, Metuchen and Princeton • NEW MEXICO, Albuquerque • NEW YORK, Albany, Buffalo (Cheektowaga), Long Island (Huntington Station), Manhattan, Rochester and Syracuse • NORTH CAROLINA, Durham/Chapel Hill • OHIO, Cleveland (Euclid), Columbus and Dayton • OKLAHOMA, Tulsa • OREGON, Eugene and Portland • PENNSYLVANIA, Allentown, Philadelphia (Bluebell) and Pittsburgh • SOUTH CAROLINA, Columbia • TENNESSEE, Knoxville and Nashville • TEXAS, Austin, Dallas and Houston • UTAH, Salt Lake City • VIRGINIA, Richmond • WASHINGTON, Bellevue • WISCONSIN, Milwaukee (Brookfield) • INTERNATIONAL—ARGENTINA, Buenos Aires • AUSTRALIA, Adelaide, Brisbane, Canberra, Melbourne, Perth and Sydney • AUSTRIA, Vienna • BELGIUM, Brussels • BOLIVIA, La Paz • BRAZIL, Rio de Janeiro and Sao Paulo • CANADA, Calgary, Edmonton, Halifax, London, Montreal, Ottawa, Toronto, Vancouver and Winnipeg • CHILE, Santiago • DENMARK, Copenhagen • FINLAND, Helsinki • FRANCE, Lyon, Grenoble and Paris • GERMAN FEDERAL REPUBLIC, Cologne, Frankfurt, Hamburg, Hannover, Munich, Nuremburg, Stuttgart and West Berlin • HONG KONG • INDIA, Bombay • INDONESIA, Djakarta • IRELAND, Dublin • ITALY, Milan, Rome and Turin • IRAN, Tehran • JAPAN, Osaka and Tokyo • MALAYSIA, Kuala Lumpur • MEXICO, Mexico City • NETHERLANDS, Utrecht • NEW ZEALAND, Auckland and Christchurch • NORWAY, Oslo • PUERTO RICO, Santurce • SINGAPORE • SPAIN, Madrid • SWEDEN, Gothenburg and Stockholm • SWITZERLAND, Geneva and Zurich • UNITED KINGDOM, Birmingham, Bristol, Epsom, Edinburgh, Leeds, Leicester, London, Manchester and Reading • VENEZUELA, Caracas •

digital

pdp11/60 processor handbook

digital equipment corporation

Additional copies of the PDP-11/60 Handbook can be ordered through Communications Services in Northborough • Order #EB06498

Copyright © 1977 by Digital Equipment Corporation

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM		

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1-1

1.1 GENERAL 1-1
1.2 PDP-11/60 HARDWARE 1-3
1.3 PDP-11/60 FEATURES 1-3
1.4 SCOPE 1-4
1.5 PERIPHERALS AND OPTIONS 1-4
I/O Devices 1-4
Storage Devices 1-5
1.6 SOFTWARE 1-5
Languages 1-6
1.7 NUMBER SYSTEMS 1-6

CHAPTER 2 SYSTEM ARCHITECTURE 2-1

2.1 INTRODUCTION 2-1
2.2 UNIBUS 2-2
2.2.1 Bidirectional Lines 2-2
2.2.2 Master-Slave Relation 2-2
2.2.3 Interlocked Communication 2-3
2.3 CENTRAL PROCESSOR 2-3
2.3.1 General Registers 2-3
2.3.2 Instruction Set 2-4
2.3.3 Processor Status Word 2-5
Processor Priority 2-5
Condition Codes 2-6
Current and Previous Mode 2-6
Trap 2-6
2.3.4 Stacks 2-6
2.4 MEMORY ORGANIZATION 2-6
2.5 AUTOMATIC PRIORITY INTERRUPTS 2-8
Direct Memory Access 2-9
Bus Requests 2-9
NPR Data Transfers 2-10
BR Transfers 2-10
Interrupt Procedure 2-10
Interrupt Servicing 2-11
Reentrant Code 2-11
Power Fail and Restart 2-11

CHAPTER 3 ADDRESSING MODES 3-1

3.1 SINGLE OPERAND ADDRESSING 3-2
3.2 DOUBLE OPERAND ADDRESSING 3-2
3.3 DIRECT ADDRESSING 3-4
3.3.1 Register Mode 3-4
3.3.2 Autoincrement Mode 3-5
3.3.3 Autodecrement Mode 3-7
3.3.4 Index Mode 3-8
3.4 DEFERRED (INDIRECT) ADDRESSING 3-10

3.5 USE OF THE PC AS A GENERAL REGISTER 3-12
3.5.1 Immediate Mode 3-13
3.5.2 Absolute Addressing 3-13
3.5.3 Relative Addressing 3-14
3.5.4 Relative Deferred Addressing 3-15
3.6 USE OF STACK POINTER AS GENERAL REGISTER 3-16
3.7 SUMMARY OF ADDRESSING MODES 3-16
3.7.1 General Register Addressing 3-16
3.7.2 Program Counter Addressing 3-18

CHAPTER 4 INSTRUCTION SET 4-1

4.1 INTRODUCTION 4-1
4.2 INSTRUCTION FORMATS 4-2
4.3 BYTE INSTRUCTIONS 4-3
4.4 SINGLE OPERAND INSTRUCTIONS 4-4
4.4.1 Single Operand Arithmetic Instructions 4-4
4.4.2 Shifts 4-11
4.4.3 Multiple Precision 4-17
4.4.4 Rotates 4-21
4.5 DOUBLE OPERAND INSTRUCTIONS 4-29
4.5.1 Double Operand General Instructions 4-29
4.5.2 Logical Instructions 4-37
4.6 PROGRAM CONTROL INSTRUCTIONS 4-41
4.6.1 Branches 4-41
4.6.2 Subroutine Instructions 4-62
4.6.3 Program Control Instructions 4-68
4.6.4 Traps 4-72
4.7 MISCELLANEOUS 4-83
4.8 CONDITION CODE OPERATORS 4-93
4.9 FLOATING POINT INSTRUCTIONS 4-94
4.10 USER CONTROL STORE INSTRUCTIONS 4-95
4.11 MAINTENANCE INSTRUCTIONS 4-96
4.11.1 Base PDP-11/60 Maintenance Instruction .. 4-97
4.11.2 FP11-E Maintenance Instructions 4-100
4.11.3 Diagnostic Instructions 4-101

CHAPTER 5 MEMORY MANAGEMENT 5-1

5.1 GENERAL 5-1
5.1.1 Memory Management 5-1
5.1.2 Programming 5-1
5.1.3 Basic Addressing 5-2
5.1.4 Active Page Registers 5-2
5.1.5 Capabilities Provided by Memory Management 5-3
5.2 RELOCATION 5-3
5.2.1 Virtual Addressing 5-3
5.2.2 Program Relocation 5-4
5.3 PROTECTION 5-6
5.3.1 Inaccessible Memory 5-6
5.3.2 Read-Only Memory 5-6
5.3.3 Multiple Address Space 5-7

5.4 ACTIVE PAGE REGISTERS 5-7
5.4.1 Page Address Registers (PAR) 5-8
5.4.2 Page Descriptor Registers (PDR) 5-8
Access Control Field (ACF) 5-9
Expansion Direction (ED) 5-9
Written Into (W) 5-11
Page Length Field (PLF) 5-11
PLF for an Upward Expandable Page 5-11
PLF for a Downward Expandable Page 5-11
5.5 VIRTUAL & PHYSICAL ADDRESSES 5-13
5.5.1 Construction of a Physical Address 5-13
5.5.2 Determining the Program Physical Address .. 5-14
5.6 STATUS REGISTERS 5-15
5.6.1 Status Register 0 (SR0) 5-15
Abort—Nonresident 5-16
Abort—Page Length 5-16
Abort—Read Only 5-16
Maintenance/Destination Mode 5-16
Mode of Operation 5-16
Page Number 5-16
Enable Relocation and Protection 5-16
5.6.2 Status Register 2 (SR2) 5-17
5.7 INSTRUCTIONS 5-17
MFPD, MFPI 5-18
MTPD, MTPI 5-19

CHAPTER 6 MEMORY 6-1

6.1 GENERAL 6-1
6.2 MEMORY SYSTEM 6-1
6.3 CACHE MEMORY 6-1
6.3.1 Introduction 6-1
6.3.2 PDP-11/60 Cache Implementation 6-3
Direct Mapping Address Mechanism 6-4
Block Size 6-4
Set Size 6-4
Write Through 6-4
6.3.3 Cache Memory Data Format 6-4
6.3.4 Physical and Cache Address 6-4
6.3.5 Processor Memory Reference 6-5
6.3.6 NPR Memory References 6-6
6.4 BACKING STORE MEMORY 6-7
6.4.1 MOS Memory 6-7
6.4.2 Core Memory 6-7
6.5 POWER FAILURE 6-7
6.6 REGISTERS 6-7
Memory System Error Register 6-8
Cache Control Register 6-9
Hit/Miss Register 6-9
6.7 PARITY 6-10
6.7.1 Basic Responses to Errors 6-10
6.7.2 System Responses to Errors 6-10

6.7.3 Cache Parity Error and Cache Control Register (CCR) 6-11

CHAPTER 7 PROGRAMMING TECHNIQUES 7-1

7.1 THE STACK 7-1
7.2 SUBROUTINE LINKAGE 7-5
7.2.1 Subroutine Calls 7-5
7.2.2 Argument Transmission 7-5
7.2.3 Subroutine Return 7-8
7.2.4 PDP-11 Subroutine Advantages 7-8
7.3 INTERRUPTS 7-9
7.3.1 General Principles 7-9
7.3.2 Nesting 7-9
7.4 REENTRANCY 7-11
7.5 POSITION INDEPENDENT CODE—PIC 7-14
7.6 CO-ROUTINES 7-15

CHAPTER 8 PROCESSOR AND COMPUTER CHARACTERISTICS 8-1

8.1 DESCRIPTION 8-1
8.2 PROCESSOR CHARACTERISTICS 8-1
8.2.1 Central Processor 8-3
8.2.2 Extended Instruction Set 8-3
8.2.3 Integral Floating Point Unit 8-3
8.2.4 Cache Memory 8-3
8.2.5 Memory Management 8-3
8.2.6 Programmable Stack Limit 8-4
8.2.7 Power Fail/Auto Restart 8-4
8.2.8 Priority Interrupt 8-5
8.2.9 UNIBUS Compatibility 8-5
8.2.10 Keypad/Numeric Display Console 8-5
8.2.11 Maintenance Features 8-6
8.2.12 FP11-E High-Speed Floating Point Processor Option 8-6
8.2.13 User Control Store, Extended Control Store and Diagnostic Control Store Options 8-6
8.3 COMPUTER CHARACTERISTICS 8-6
8.3.1 M9302 UNIBUS Terminator 8-7
8.3.2 M9301-H Bootstrap Loader/Self Diagnostic/ Terminator 8-7
8.3.3 DL11-W Serial Line Unit/Line Frequency Clock 8-8
8.3.4 UNIBUS Memories 8-8
8.3.5 Basic PDP-11/60 System 8-9
8.3.6 Computer System Specifications 8-10
8.3.7 Expansion Capability 8-10

CHAPTER 9 USER MICROPROGRAMMING 9-1

9.1 INTRODUCTION 9-1
9.1.1 Microprogram Control 9-2

9.1.2 Dynamic Control Structure 9-2
9.1.3 Simple Datapath 9-2
9.1.4 Example of Microprogramming 9-4
9.1.5 Microprogramming vs. Macroprogramming .. 9-5
9.2 OPTIONS 9-5
9.2.1 User Control Store 9-5
9.2.2 Extended Control Store 9-6
9.2.3 Diagnostic Control Store 9-6
9.2.4 Applications 9-6
9.3 PDP-11/60 IMPLEMENTATION 9-7
9.3.1 Control Store Address Space 9-7
9.3.2 Local Store Flexibility 9-7
9.3.3 Loading/Checking User Control Store 9-8
9.3.4 Macro Datapath 9-9
ALU (Arithmetic Logic Unit) 9-9
ASP (A Scratchpad) 9-10
BSP (B Scratchpad) 9-10
CSP (C Scratchpad) 9-10
D Register (Data Register) 9-10
SR (Shift Register) 9-10
Shift Tree 9-11
CNTR (Counter) 9-11
BA (Bus Address) 9-11
Interface to the Outside 9-11
9.3.5 Microword 9-11
UWORD Functions in a Normal Transfer 9-11
9.3.6 Microprogrammability 9-13
High Speed Registers 9-13
Subroutine and Branching 9-13
Emit Constant 9-14
Intraprocessor Control 9-14
9.3.7 Macro Access to Micro Control 9-14

CHAPTER 10 FLOATING POINT PROCESSOR 10-1

10.1 INTRODUCTION 10-1
10.2 OPERATION 10-1
10.3 ARCHITECTURE 10-2
10.4 FLOATING POINT DATA FORMATS 10-4
10.4.1 Non-vanishing Floating Point Numbers 10-4
10.4.2 Floating Point Zero 10-4
10.4.3 The Undefined Variable 10-4
10.4.4 Floating Point Data 10-5
10.5 FLOATING POINT UNIT STATUS REGISTER (FPS Register) 10-6
10.6 FLOATING EXCEPTION CODE AND ADDRESS REGISTER 10-10
10.7 FLOATING POINT PROCESSOR INSTRUCTION ADDRESSING 10-10
10.8 ACCURACY 10-11
10.9 FLOATING POINT INSTRUCTIONS 10-12

(See Appendix C for Instruction Index)
Floating Point Instruction Format 10-13
General Definitions 10-13

CHAPTER 11 PROGRAMMERS' CONSOLE RELIABILITY AND MAINTENANCE 11-1

11.1 INTRODUCTION—CONSOLE 11-1
11.2 KY11-P PROGRAMMERS' CONSOLE 11-1
11.2.1 Power-up 11-1
11.2.2 Starting and Stopping 11-3
11.2.3 Console Indicators and Switches 11-3
11.2.4 Console Internal Registers 11-4
11.2.5 Functions of Switches and Indicators 11-4
11.2.6 Console Mode Functions 11-6
11.3 INTRODUCTION—RELIABILITY AND MAINTENANCE 11-11
11.4 RELIABILITY 11-11
11.4.1 Hardware Features 11-12
11.4.2 Fault Tolerance 11-13
Confidence Check Features 11-13
M9301-YH Diagnostic Bootstrap Loader 11-13
Base PDP-11/60 Power-up Tests 11-14
Parity Check During System Operation 11-14
Fault Tolerance Through Error Logging 11-14
Fault Tolerance with Automatic Back-up on Cache Errors 11-14
Fault Tolerance with Cache Turn-off 11-15
Fault Tolerance with Integral Floating Point Instruction Set 11-15
Fault Tolerance with User Control Store 11-15
11.5 MAINTENANCE 11-15
11.5.1 PDP-11/60 Microdiagnostics 11-17
11.5.2 KU 116-BB Diagnostic Control Store 11-17
11.5.3 Maintenance Features of the System Package 11-19
11.5.4 Maintenance Features of the Console 11-20
11.5.5 Maintenance Through Software and Firmware 11-20

APPENDIX A UNIBUS ADDRESSES A-1

A.1 INTERRUPT & TRAP VECTORS A-1
A.2 FLOATING VECTORS A-2
A.3 FLOATING ADDRESSES A-3

APPENDIX B PDP-11/60 INSTRUCTION TIMING B-1

B.1 PDP-11/60 INSTRUCTION EXECUTION TIME B-1
B.2 BASIC INSTRUCTION SET TIMING B-1
B.2.1 Using the Chart Times B-1

B.2.2 Chart Times B-2
B.3 FLOATING POINT INSTRUCTION TIMING B-6

APPENDIX C INSTRUCTION INDEX C-1

Floating Point Instruction Set C-2
Single Operand Instructions C-2
Double Operand Instructions C-2

FIGURES

Figure	1-1	PDP-11/60 System Block Diagram	1-2
Figure	2-1	PDP-11/60 System Simplified Block Diagram	2-2
Figure	2-2	The General Registers	2-4
Figure	2-3	Processor Status Word	2-5
Figure	2-4	Memory Addresses	2-7
Figure	2-5	High & Low Bytes	2-7
Figure	2-6	Word and Byte Addresses	2-7
Figure	2-7	UNIBUS Priority	2-8
Figure	5-1	Active Page Registers	5-3
Figure	5-2	Simplified Memory Relocation Example	5-4
Figure	5-3	Relocation of a 32K Word Program into 124K Word Physical Memory	5-6
Figure	5-4	Page Address Register	5-8
Figure	5-5	Page Descriptor Register	5-9
Figure	5-6	Example of an Upward Expandable Page	5-10
Figure	5-7	Example of a Downward Expandable Page	5-12
Figure	5-8	Interpretation of a Virtual Address	5-13
Figure	5-9	Displacement Field of Virtual Address	5-13
Figure	5-10	Construction of a Physical Address	5-14
Figure	5-11	Format of Status Register #0 (SR0)	5-15
Figure	5-12	Format of Status Register 2 (SR2)	5-17
Figure	6-1	Simplified PDP-11/60 System	6-2
Figure	6-2	Cache Memory System Relationship	6-3
Figure	6-3	Cache Memory Data Format	6-5
Figure	6-4	Physical Address Format	6-5
Figure	7-1	Stack Addresses	7-1
Figure	7-2	Word and Byte Stacks	7-2
Figure	7-3	Illustration of Push and Pop Operations	7-3
Figure	7-4	Register Saving Without the Stack	7-3
Figure	7-5	Register Saving Using the Stack	7-4
Figure	7-6	Byte Stack Used as a Character Buffer	7-4
Figure	7-7	JSR Using R5	7-5
Figure	7-8	JSR	7-5
Figure	7-9	Argument Transmission—Register Autoincrement Mode	7-6
Figure	7-10	Argument Transmission—Register Autoincrement Deferred Mode	7-6
Figure	7-11	Transmitting Stacks as Arguments	7-7
Figure	7-12	Shifting Indexed Base	7-7
Figure	7-13	Constant Index Base using "R6 Copy"	7-8
Figure	7-14	Nested Interrupt Service Routines and Subroutines	7-11
Figure	7-15	Reentrant Routines	7-12
Figure	7-16	Reentrant Routine Sharing	7-12
Figure	7-17	Multiple R6 Stack	7-14
Figure	7-18	Co-Routine Interaction	7-15
Figure	8-1	PDP-11/60 Processor Backplane	8-2
Figure	8-2	PDP-11/60 System with RK06 Disks	8-9
Figure	8-3	PDP-11S60 Configuration	8-11
Figure	8-4	PDP-11T60 Configuration	8-12
Figure	9-1	Sample Datapath (not PDP-11/60)	9-3

Figure 9-2 Microinstruction Fields 9-5
Figure 9-3 Microcode Address Space 9-8
Figure 9-4 Simplified Block Diagram of PDP-11/60 Datapath 9-9
Figure 9-5 Simplified and Detail Representation of the PDP-11/60 Microword 9-12
Figure 10-1 Floating Point Processor and Central Processor of the PDP-11/60 10-3
Figure 11-1 PDP-11/60 Console 11-2
Figure B-1 Load Class of Instructions B-11
Figure B-2 Load Class of Instructions (Integer Conversion) B-12
Figure B-3 Store Class of Instructions B-12
Figure B-4 Store Class of Instructions (Integer Convert) B-13
Figure B-5 Calculation of Effective Times for Load Class Instructions B-15

TABLES

Table	5-1	PAR/PDR Address Assignments	5-8
Table	5-2	Access Control Field Keys	5-9
Table	5-3	Relating Virtual Address to PAR/PDR Set	5-15
Table	6-1	PDP-11/60 Cache Implementation	6-4
Table	6-2	Hit or Miss Operations	6-6
Table	6-3	Actions upon Cache Parity	6-13
Table	8-1	Expansion Power	8-13
Table	9-1	Control Signals Status	9-4
Table	9-2	ALU Control Field Encoding	9-10
Table	11-1	Internal Registers Read/Write Function Codes	11-8
Table	11-2	Processor Error Log	11-18
Table	B-1	Source Address Time	B-2
Table	B-2	Destination Address Time	B-3
Table	B-3	Execute, Fetch Time (Double Operand)	B-3
Table	B-4	Execute, Fetch Time (Single Operand)	B-4
Table	B-5	Branch Instructions	B-4
Table	B-6	JUMP Instructions	B-5
Table	B-7	Miscellaneous Instructions	B-5
Table	B-8	EIS Instructions MUL, DIV, ASH, ASHC	B-6
Table	B-9	Definitions	B-7
Table	B-10	Floating Point Instructions	B-8
Table	B-11	Address Calculation (Address/Double)	B-9
Table	B-12	Address Calculation (Integer)	B-9
Table	B-13	Shared Execution Time	B-10
Table	B-14	FP11-E Execution Times	B-10

CHAPTER 1

INTRODUCTION

1.1 GENERAL

The PDP-11/60, as a mid-range PDP-11 computer, features significant standard system capabilities and maintenance features. It has a favorable performance/price ratio for scientific, business and industrial applications. Through a microprogramming capability, the PDP-11/60 contains a new level of customer flexibility and expansion. A number of large-computer maintenance features are introduced into this minicomputer to insure high integrity and availability. The PDP-11/60 is oriented toward high-performance real-time applications and medium performance multi-user, multi-task timeshared applications. Memory expands to 256K bytes. Memory relocation and protection are built in. Hardware Multiply, Divide, and 32- and 64-bit single- and double-precision floating point arithmetic are standard.

Systems, including the Central Processor Unit (CPU), memory mass storage, and other options are housed in a single low-profile cabinet. A bootstrap loader and an octal display console are provided to facilitate operator control of the computer.

High speed and throughput have been designed into the PDP-11/60. For example, register-to-register operations can be performed in as little as 340 nanoseconds. A 2048-byte high-speed cache memory is standard and maintains a copy of selected portions of the slower, less expensive, main memory; thereby providing faster access to instructions and data. This results in effective memory cycle times of less than 550 ns.

A portion of the control store is available for customer use in the PDP-11/60. 2.5K words of this address space are reserved for the base machine. The remaining 1.5K words of address space are used for three immediately available firmware products offering a high degree of flexibility and providing for future enhancements. Two of the products are User Control Store (UCS—1K address space) and Extended Control Store (ECS—1.5K address space). These products are general purpose options for use in the development of specific microcode routines for critical applications. The third product is Diagnostic Control Store (DCS —2K address space) and represents a specific, hardware supported application of ECS. DCS is a fault isolator which facilitates fault isolation in the central processor to the module level.

The PDP-11/60 machine contains a basic floating point unit which implements floating point instructions. This instruction set (32- and 64-bit formats) is standard and allows the use of the optimizing FORTRAN IV–PLUS compiler in the base computer. For heavy scientific loads, a high-speed floating point processor option (FP11-E) is available which also implements these instructions, and provides performance improvement over the integral floating point unit.

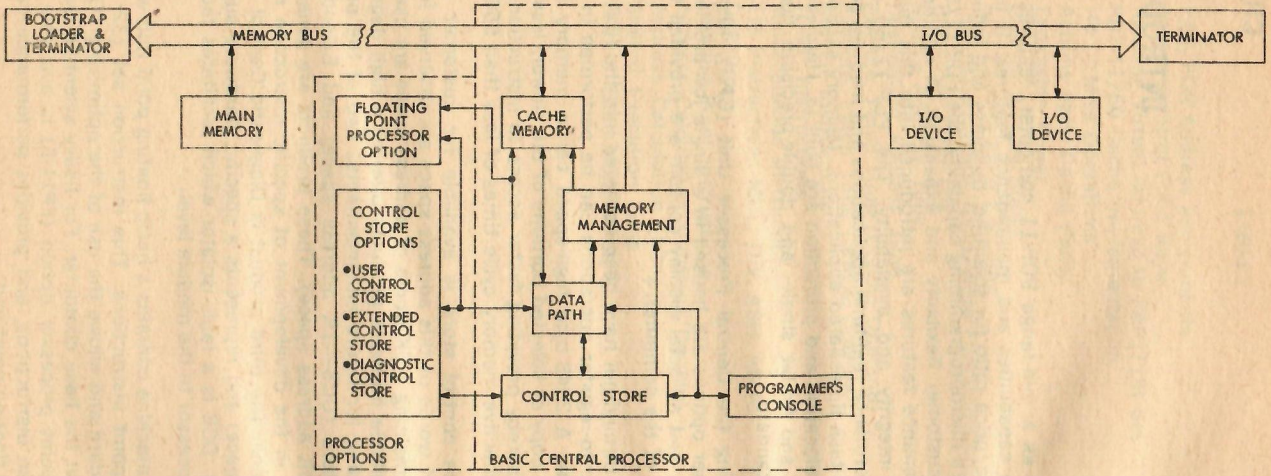


Figure 1-1 PDP-11/60 System Block Diagram

An in-depth Reliability And Maintainability Program (RAMP) provides the PDP-11/60 with many integrity and maintainability features. Parity and error-correcting memory is offered. Error log software and hardware anticipate system degradation and allow field service personnel to be called prior to 'system outages'. Self-diagnosing equipment coupled with specific fault isolation diagnostics decrease the fault isolation time.

1.2 PDP-11/60 HARDWARE

The following equipment is included as an integral part of the basic PDP-11/60 computer:

- KD11-K Processor
- 2K byte Cache memory (K = 1024)
- Extended Instruction Set (EIS)—MUL, DIV, ASH, ASHC
- Floating Point Instruction Set (FP11-C equivalent instructions)
- Power Fail/Auto Restart
- Hardware Memory Management (KT11-D equivalent)
- Memory in 64K byte increments of ECC MOS or parity core
- Battery Backup for MOS versions
- Line Clock and Terminal Controller (DL11-W)
- Multi-device Bootstrap Loader/Self-Diagnostic Test
- Keypad/Numeric Display Console
- Logic Cage (BA11 P) with Power Supply (H7420) and Regulators

The following optional PDP-11/60 equipment is available:

- Floating Point Processor (FP11-E)
- Diagnostic Control Store (Fault Isolator), KU116-BB
- User Control Store (Read/write control store extension), KU116-AA
- Extended Control Store (Read-only control store extension), KU116-AB

1.3 PDP-11/60 FEATURES

The PDP-11/60 computer, as a member of the PDP-11 family, has the following features:

- Single and double operand instructions that provide a powerful and convenient set of programming instructions
- 16-bit word (two 8-bit bytes) with direct addressing of 32K words or 64K bytes
- Parity detection on each 8-bit core memory byte
- Error detection and correction (ECC) on MOS memory
- Hardware address expansion and protection allowing memory addressing to 124K words
- Word or byte processing—very efficient handling of 8-bit data without the need to rotate, swap, or mask
- Asynchronous operation—system components run at their highest possible speed; replacement with faster subsystems means faster operation without other hardware or software changes

- Modular component design—extreme ease and flexibility in configuring systems
- Stack processing—hardware sequential memory manipulation makes it easy to handle structured data, subroutines, and interrupts
- Direct Memory Access (DMA)—inherent in the architecture is direct memory access for multiple devices
- 8 programmable general-purpose registers—used interchangeably for accumulators or address generation
- 6 64-bit Floating Point internal registers
- 64-bit wide data path for FP11-E Floating Point option
- Automatic Priority Interrupt—four-line, multi-level system permits groupings of interrupt lines according to response requirements
- Vectored interrupts—fast interrupt response without device polling
- Power Fail/Automatic Restart—hardware detection and software protection for fluctuations in the ac power line

1.4 SCOPE

This handbook describes the PDP-11/60 computer designed and manufactured by Digital Equipment Corporation.

The intent is to provide information on operation of the computer; in general, performance, features, and basic programming. This handbook is not intended to be the sole reference for the computer. More comprehensive and detailed information is available in the PDP-11/60 Maintenance and Programming Manuals and the UCS Microprogrammers' Guide.

1.5 PERIPHERALS AND OPTIONS

Digital Equipment Corporation designs and manufactures many of the peripheral devices offered with PDP-11s. As a designer and manufacturer of peripherals, DIGITAL can offer extremely reliable equipment, lower prices, quantity discounts, and a wide range of computer options to choose from.

I/O Devices

All PDP-11 systems can use ASCII terminals as the basic I/O device. However, I/O capabilities can be increased with disk cartridges, disk packs, magnetic tape drives, alphanumeric display devices and high-performance terminals.

The LA36 DECwriter II 30cps keyboard printer, a totally DIGITAL-designed and built teleprinter, has several advantages over standard electromechanical typewriter terminals, including higher speed, fewer mechanical parts, and very quiet operation.

PDP-11 devices include:

- Floppy disk, RX01
- VT52 DECterminal II alphanumeric display
- Tape Drive, TS03
- Tape Transport, TU10, TU16, TU45

- Diskpack drive and control unit, RJP04, RJP05, and RJP06
- Disk drive with removable cartridge, RK05J
- Disk drive with non-removable cartridge, RK05F
- Disk drive with removable cartridge, RK06
- High-speed line printers, LA180, LP11, LV11
- Card readers, CR11, CD11, CM11
- Graphics terminal, GT40
- Synchronous and Asynchronous Single and Multi-channel Communications Interfaces
- DMC-11 Microprocessor

Storage Devices
Storage devices range from paper tape to mass storage magnetic tapes, disks, and disk packs. The disk packs, in addition to providing mass storage capability, provide the additional benefit of serving as I/O devices for use in transferring programs. With the UNIBUS, a large number of storage devices, in any combination, may be connected to a PDP 11 system. For applications which require handling of large amounts of data, DIGITAL offers the industry-compatible TU16 Magtape. Disk storage includes fixed-head disk units and moving-head removable cartridge and disk pack units, such as the RP04 Disk Pack which can store up to 44 million words.

1.6 SOFTWARE

The PDP-11 family of central processors, memories, and peripherals, is supported by a comprehensive family of licensed software products. This software family includes support for small stand-alone configurations, disk-based real-time and program development systems, large multiprogramming and timesharing systems, and many diverse dedicated applications. Some examples of general-purpose operating systems and standard high-level language processors are:

- MULTI-TASKING PROCESS CONTROL SYSTEM (RSX-11M)—An effcient multi-tasking system suitable for controlling many processes simultaneously, in a protected environment with concurrent development of new programs. Utilities include a MACRO assembler, task builder (linker), editor, debugger, and file utility programs.
- MULTI-USER REAL-TIME OPERATING SYSTEM (RSX-11D)—a large multi-user operating system for concurrent real-time applications execution, program development, and general data processing. Tasks to run in the priority-scheduled, event-driven, multiprogramming system may be written in MACRO assembly language, FORTRAN IV, FORTRAN IV–PLUS or COBOL. The system features task accounting and reporting for system management, on-line device error logging and analysis for system maintenance, and batch stream processing for low-priority tasks.
- SINGLE-USER ON-LINE PROGRAM DEVELOPMENT SYSTEM (RT-11) —a small, powerful, easy-to-use disk-based system for program development of fast on-line (real-time) applications. A Foreground/Background version can accommodate simultaneous program development

in the background with real-time applications in the foreground. A MACRO assembler, linker, editor, debugger, and file utility programs are included.

Languages
PDP-11 FORTRAN IV-PLUS—an optional language processing system for RSX-11D and RSX-11M. The FORTRAN IV-PLUS language conforms to the specifications for the American National Standard FORTRAN X39-1966, is upward-compatible with the FORTRAN IV compiler and supports the same enhancements to the ANSI standard. FORTRAN IV-PLUS is an optimizing compiler which creates code that directly references the FP11-E Floating Point Processor option or the integral floating point to provide optimum speed of floating point instructions.

PDP-11 FORTRAN IV—an extended version of ANSI-standard FORTRAN is available with RSX-11M and RSX-11D, and available under RT-11. FORTRAN IV is designed for fast compilation yet requires very little main memory. Under RT-11, FORTRAN IV features the same signal-processing and graphics extensions as BASIC-11.

BASIC-11—an extended version of Dartmouth, Standard BASIC is available for PTS-11 and RT-11. Many applications, such as signal processing and graphics, are accessed by the user through extensions to this simple, yet powerful, language. A multiuser version is available under PTS-11 and RT-11.

PDP-11 COBOL—to supplement the business data processing needs often associated with large-scale PDP-11 system applications, an ANSI-74 COBOL language is available under RSX-11M and RSX-11D. Running as a BATCH job, COBOL enhances the RSX-11M and RSX-11D total job computing system, where some business data processing is required.

In addition to the above-mentioned general-purpose licensed software products, DIGITAL offers a great number of optional and applications-oriented products. A wide range of educational, consulting, and maintenance services are also offered to ensure full utility of any PDP-11 system. For a complete and detailed listing of DIGITAL software products and services, consult the latest CATALOG OF SOFTWARE PRODUCTS AND SERVICES.

1.7 NUMBER SYSTEMS

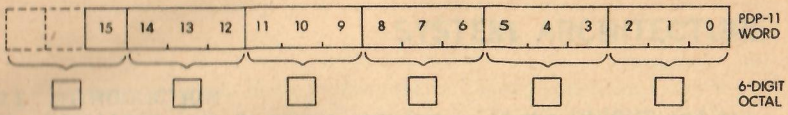
Throughout this handbook, three number systems will be used: octal, binary, and decimal. So as not to clutter all numbers with subscripted bases, the following general convention will be used.

Octal—for address locations, contents of addresses, and operation codes for instruction; in most cases there will be words of 6 octal digits.

Binary—for describing a single binary element; when referring to a PDP-11 word, it will be 16 bits long.

Decimal—for all normal references to quantities.

Octal Representation



The 16-bit PDP-11 word can be represented conveniently as a 6-digit octal word. Bit 15, the Most Significant Bit (MSB), is used directly as the Most Significant Digit of the octal word. The other 5 octal digits are formed from the corresponding groups of 3 bits in the binary word.

When an extended address of 18 bits is used (shown later in the handbook), the Most Significant Digit of the octal word is formed from bits 17, 16, and 15. For unsigned numbers, the correspondence between decimal and octal is:

Decimal	Octal	
0	000000	
$(2^{16} - 1) =$ 65,535	177777	(16-bit limit)
$(2^{18} - 1) =$ 262,143	777777	(18-bit limit)

CHAPTER 2

SYSTEM ARCHITECTURE

2.1 INTRODUCTION

The PDP-11/60, with its unique combination of the UNIBUS and processor cache memory, allows I/O transfers to memory to occur simultaneously with central processor (CP) accesses from cache memory. I/O transfers do not cause the CP to halt, and expensive multiport memories, with a more complex bus structure, are not necessary. The cache/UNIBUS arrangement results in a system-oriented computer that can handle high I/O rates without significant degradation of processor speed.

Since the cache memory is part of the processor, the usual operation of UNIBUS I/O devices and of UNIBUS memories is unchanged. Standard PDP-11 UNIBUS operations occur for the UNIBUS, central processor, memory organization, and automatic priority interrupts; and are noted in this chapter. UNIBUS addresses are listed in Appendix A.

The central processor is located between I/O devices and memories to insure proper cache invalidation upon I/O references to memory. This invalidation is automatic and occurs within the cache control of the central processor. (See Chapter 6.) Cache memory operation, in conjunction with bus ownership in the PDP-11/60, allows the elimination of DATIP UNIBUS operations for most instructions at most addresses. DATI operations are used for DATIP operations in instruction execution with the exception of I/O page references, and for all addresses of the Arithmetic Shift Right Byte (ASRB) instruction. (See Chapter 4.) These changes do not affect program operation, and maintain compatability with existing UNIBUS devices.

Other elements of PDP-11 systems such as the instructions, addressing modes and programming techniques remain unchanged, and are noted in Chapters 4, 3, and 7, respectively. The processor of the PDP-11/60, through its extensive microcoded capability, has expanded the basic instruction set. The Floating Point Instructions of Chapter 10 are implemented within the basic machine. A faster implementation of this instruction also exists with an add-on FP11-E Floating Point Processor described in Chapter 10.

Memory Management logic provides memory addressing capability for 256K bytes in two modes of operation: Kernel and User. Operational specifications for this standard capability are listed in Chapter 5. The memory system itself includes a bipolar cache memory. The structure of this memory with its associated controls and error detection is discussed in Chapter 6. Extra control and error detection exist in the central processor. These processor features and their interaction upon other PDP-11 and UNIBUS elements of the PDP-11/60 are described in Chapter 8. A new feature of the PDP-11/60 processor is USER Microprogramming. This option provides access to the basic control structure of the machine and allows direct manipulation of the processor. Introductory information upon its application and implementation is presented in Chapter 9.

The console of the PDP-11/60 is standard and provides an operator interface to the central processor. Program and system initialization, control, and analysis are possible. Since the console is a strong tool in diagnostic efforts, Chapter 11 also contains information upon the overall Reliability And Maintainability Program (RAMP) for the PDP-11/60 system.

2.2 UNIBUS

System components connect to and communicate with each other on a single high-speed bus known as the UNIBUS—a key to the PDP-11's many strengths. Addresses, data, and control information are sent along the 56 lines of the bus.

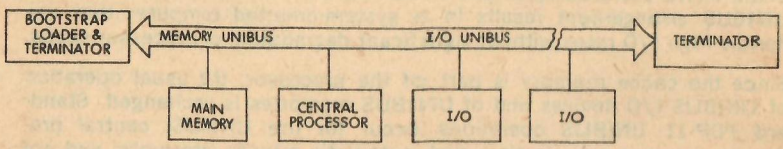


Figure 2.1 PDP-11/60 System Simplified Block Diagram

The form of communication is the same for every device on the UNIBUS. The processor uses the same set of signals to communicate with memory as with peripheral devices. Peripheral devices also use this set of signals when communicating with the processor, memory or other peripheral devices. Each device, including memory locations, processor registers, and peripheral device registers, is assigned an address on the UNIBUS. Thus, peripheral device registers may be manipulated as flexibly as core memory by the central processor. All the instructions that can be applied to data in core memory can be applied equally well to data in peripheral device registers. This is an especially powerful feature, considering the special capability of PDP-11 instructions to process data in any memory location as though it were an accumulator.

2.2.1 Bidirectional Lines

With bidirectional and asynchronous communications on the UNIBUS, devices can send, receive, and exchange data independently without processor intervention. For example, a cathode ray tube (CRT) display can refresh itself from a disk file while the central processor unit (CPU) attends to other tasks. Because it is asynchronous, the UNIBUS is compatible with devices operating over a wide range of speeds.

2.2.2 Master-Slave Relation

Communication between two devices on the bus is in the form of a master-slave relationship. At any point in time, there is one device that has control of the bus. This controlling device is termed the "bus master." The master device controls the bus when communicating with another device on the bus, termed the "slave." A typical example of this relationship is the processor, as master, fetching an instruction from memory (which is always a slave). Another example is the disk, as master, transferring data to memory, as slave. Master-slave relationships are dynamic. The processor, for example, may pass bus control

to a disk. The disk, as master, could then communicate with a slave memory bank.

Since the UNIBUS is used by the processor and all I/O devices, there is a priority structure to determine which device gets control of the bus. Every device on the UNIBUS which is capable of becoming bus master is assigned a priority. When two devices which are capable of becoming a bus master request use of the bus simultaneously, the device with the higher priority will receive control.

2.2.3 Interlocked Communication

Communication on the UNIBUS is interlocked so that for each control signal issued by the master device, there must be a response from the slave in order to complete the transfer. Therefore, communication is independent of the physical bus length (as far as timing is concerned) and the timing of each transfer is dependent only upon the response time of the master and slave devices. The asynchronous operation precludes the need for synchronizing with, and waiting for, clock impulses. Thus, each system is allowed to operate at its maximum possible speed.

Input/output devices transferring directly to or from memory are given highest priority and may request bus mastership and steal bus and memory cycles during instruction operations. The processor resumes operation immediately after the memory transfer. Multiple devices can operate simultaneously at maximum direct memory access (DMA) rates by "stealing" bus cycles.

Full 16-bit words or 8-bit bytes of information can be transferred on the bus between a master and a slave. The information can be instructions, addresses, or data. This type of operation occurs when the processor, as master, is fetching instructions, operands, and data from memory, and storing the results into memory after execution of instructions. Direct data transfers occur between a peripheral device control and memory.

2.3 CENTRAL PROCESSOR

The central processor, connected to the UNIBUS as a subsystem, controls the time allocation of the UNIBUS for peripherals and performs arithmetic and logic operations and instruction decoding. It contains multiple high-speed general-purpose registers which can be used as accumulators, address pointers, index registers, and other specialized functions. The processor can allow data transfers directly between I/O devices and memory without disturbing the processor registers, does both single- and double-operand addressing, and handles both 16-bit word and 8-bit byte data. Processor information of a general and usual nature for PDP-11 systems is within this section; specific and expanded processor features for the PDP-11/60 are defined in Chapters 6, and 8 thru 11.

2.3.1 General Registers

The central processor contains 8 general registers which can be used for a variety of purposes. The registers can be used as accumulators, index registers, autoincrement registers, autodecrement registers, or as stack pointers for temporary storage of data. Chapter 3 on Addressing describes these uses of the general registers in more detail. Arithmetic operations can be from one general register to another, from one memory or device register to another, or between memory or a device register and a general register. Refer to Figure 2-2.

R7 is used as the machine's Program Counter (PC) and contains the address of the next instruction to be executed. It is a general register normally used only for addressing purposes and not as an accumulator for arithmetic operations.

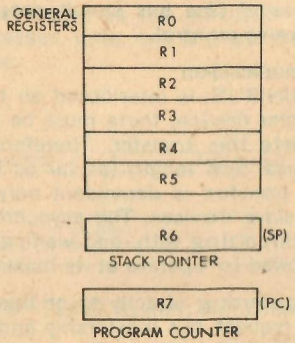


Figure 2-2 The General Registers

R6 is normally used as the Stack Pointer indicating the last entry in the appropriate stack (a common temporary storage area with "last-in first-out" characteristics). The Memory Management feature provides 2 separate stack pointers, 1 for each of 2 operating modes (Kernel and User).

2.3.2 Instruction Set

The instruction complement uses the flexibility of the general-purpose registers to provide over 400 powerful hard-wired instructions—the most comprehensive and powerful instruction repertoire of any computer in the 16-bit class. Unlike conventional 16-bit computers, which usually have three classes of instructions (memory reference instructions, operate or AC control instructions, and I/O instructions), all operations in the PDP-11 are accomplished with one set of instructions. Since peripheral device registers can be manipulated as flexibly as core memory by the central processor, instructions that are used to manipulate data in core memory may be used equally well for data in peripheral device registers. For example, data in an external device register can be tested or modified directly by the CPU, without bringing it into memory or disturbing the general registers. One can add data directly to a peripheral device register, or compare logically or arithmetically. Thus all PDP-11 instructions can be used to create a new dimension in the treatment of computer I/O and the need for a special class of I/O instructions is eliminated.

The basic order code of the PDP-11 uses both single- and double-operand address instructions for words or bytes. The PDP-11 therefore performs very efficiently, in one step, such operations as adding or subtracting two operands, or moving an operand from one location to another.

PDP-11 Approach

```
ADD A,B          ;add contents of location A to location B, store results at location B
```

Conventional Approach

```
LDA A            ;load contents of memory location A into AC
ADD B            ;add contents of memory location B to AC
STA B            ;store result at location B
```

Addressing

Much of the power of the PDP-11 is derived from its wide range of addressing capabilities. PDP-11 addressing modes include sequential addressing forwards or backwards, addressing indexing, indirect addressing, 16-bit word addressing, 8-bit byte addressing, and stack addressing. Variable length instruction formating allows a minimum number of bits to be used for each addressing mode. This results in efficient use of program storage space.

2.3.3 Processor Status Word

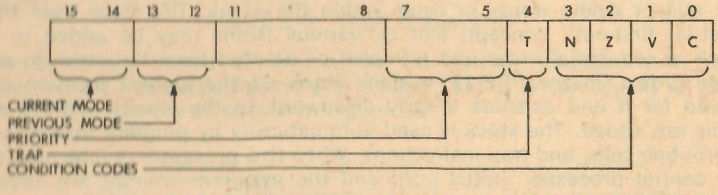


Figure 2-3 Processor Status Word

The Processor Status Word (PS), at location 777776, contains information on the current status of the PDP-11. This information includes the current processor priority: current and previous modes; the condition codes describing the results of the last instruction; and an indicator for detecting the execution of an instruction to be trapped during program debugging.

Processor Priority

The central processor operates at any one of eight levels of priority, 0-7. When the CPU is operating at level 7 an external device cannot interrupt it with a request for service. The central processor must be operating at a lower priority than the external device's request in order for the interruption to take effect. The current priority is maintained in the Processor Status Word (bits 5-7). The 8 processor levels enable high-speed devices requiring rapid servicing to take priority over, and disable interruption by, low-speed devices.

Condition Codes
The condition codes contain information on the result of the last CPU operation.

The bits are set as follows:

Z = 1, if the result was zero
N = 1, if the result was negative
C = 1, if the operation resulted in a carry from the MSB
V = 1, if the operation resulted in an arithmetic overflow

Current and Previous Mode
These bits indicate the relocation and protection mode of the Memory Management unit at the present time (current) and prior to the last mode change (previous). Two modes are implemented in the PDP-11/60, referred to as Kernel and User modes. Another mode, Supervisor, is not implemented. If Supervisor mode is attempted, the PDP-11/60 will default (without a macrotrap) to Kernel mode. See Chapter 5 for details on Memory Management operation.

Trap
The trap bit (T) can be set or cleared under program control. When set, a processor trap will occur through location 14 on completion of instruction execution and a new Processor Status Word will be loaded. This bit is especially useful for debugging programs as it provides an efficient method of installing breakpoints.

2.3.4 Stacks
In the PDP-11, a stack is a temporary data storage area which allows a program to make efficient use of frequently accessed data. A program can add or delete words or bytes within the stack. The stack uses the "last-in, first-out" concept; that is, various items may be added to a stack in sequential order and retrieved or deleted from the stack in reverse order. On the PDP-11, a stack starts at the highest location reserved for it and expands linearly downward to the lowest address as items are added. The stack is used automatically by program interrupts, subroutine calls, and trap instructions. When the processor is interrupted, the central processor status word and the program counter are saved (pushed) onto the stack area. A new status word and program counter are then automatically acquired from an area in memory which is reserved for interrupt service routine pointers (vector area). A return from the interrupt instruction restores the original processor status and program counter and returns to the interrupted program without software intervention.

2.4 MEMORY ORGANIZATION
A memory can be viewed as a series of locations, with a number (address) assigned to each location. Thus 16,384 bytes of PDP-11 memory could be shown as in Figure 2-4.

Because PDP-11 memories are designed to accommodate both 16-bit words and 8-bit bytes, the total number of addresses does not correspond to the number of words but to the number of bytes. An 8K-word memory contains 16K bytes and consists of 037777 octal locations. Words always start at even-numbered locations.

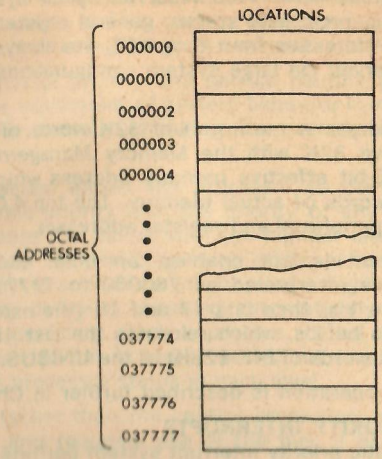


Figure 2-4 Memory Addresses

A PDP-11 word is divided into a high byte and a low byte as shown in Figure 2-5.

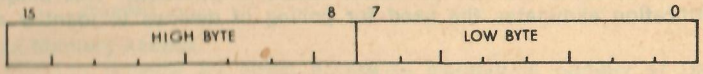


Figure 2-5 High & Low Byte

Low bytes are stored at even-numbered memory locations and high bytes at odd-numbered memory locations. Thus it is convenient to view the PDP-11 memory as shown in Figure 2-6.

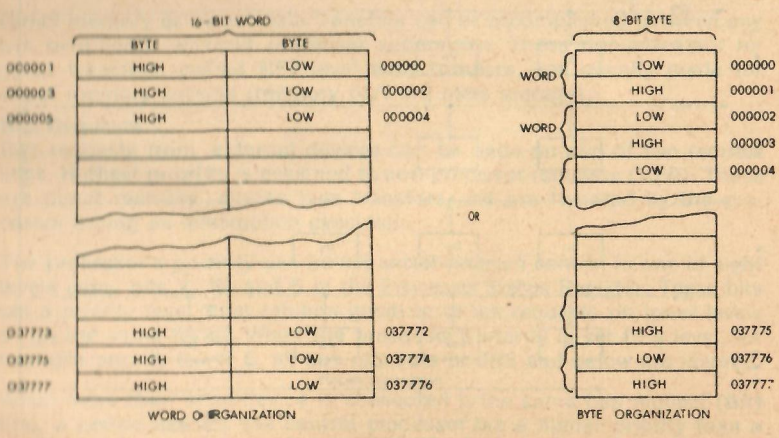


Figure 2-6 Word and Byte Addresses

Certain memory locations have been reserved by the system for interrupt and trap handling, processor stacks, general registers, and peripheral device registers. Addresses from 0 to 370_8 are always reserved and those to 777_8 are reserved on large system configurations for traps and interrupt handling.

A 16-bit word can address a maximum of 32K words of memory. The user can expand above 32K with the Memory Management unit. This device provides an 18-bit effective memory address which permits addressing up to 124K words of actual memory. The top 4,096 word locations are reserved for peripheral and register addresses.

If Memory Management is not enabled, an octal address between 160000 and 177777 is interpreted as 760000 to 777777. That is, if bits 15, 14 and 13 are 1's, then bits 17 and 16 (the extended address bits) are considered to be 1's, which relocates the last 4K words (28K-32K) to the highest 4K words (124K-128K) of the UNIBUS.

Memory Management operation is described further in Chapter 5.

2.5 AUTOMATIC PRIORITY INTERRUPTS

The multi-level automatic priority interrupt system permits the processor to respond automatically to conditions outside the system. Any number of separate devices can be attached to each level.

Each peripheral device in the PDP-11 system has a pointer to its own pair of memory words (one points to the device's service routine, and the other contains the new processor status information). This unique identification eliminates the need for polling of devices to identify an

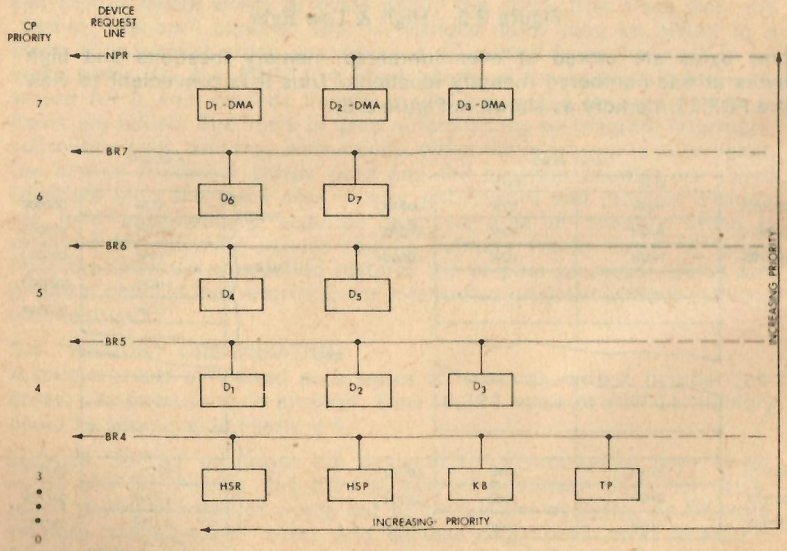


Figure 2-7 UNIBUS Priority

interrupt, since the interrupt service hardware selects and begins executing the appropriate service routine after having automatically saved the status of the interrupted program segment.

The device's interrupt priority and service routine priority are independent. This allows adjustment of system behavior in response to real-time conditions, by dynamically changing the priority level of the service routine.

The interrupt system allows the processor to continually compare its own programmable priority with the priority of any interrupting devices and to acknowledge the device with the highest level above the processor's priority level. The servicing of an interrupt for a device can be interrupted in order to service an interrupt of a higher priority. Service to the lower priority device is resumed automatically upon completion of the higher level servicing. Such a process, called nested interrupt servicing, can be carried out to any level without requiring the software to save and restore processor status at each level.

When a device (other than the central processor) is capable of becoming bus master and requests use of the bus, it is generally for one of two purposes:

1. To make a non-processor transfer of data directly to or from memory
2. To interrupt program execution and force the processor to go to a specific address where an interrupt service routine is located.

Direct Memory Access

All PDP-11s provide for direct access to memory. A number of DMA devices may be attached to the UNIBUS. Maximum priority is given to DMA devices, thus allowing memory data storage or retrieval at memory cycle speeds. Response time is minimized by the organization and logic of the UNIBUS, which samples requests and priorities in parallel with data transfers.

Direct memory or direct data transfers can be accomplished between any two peripherals without processor supervision. These non-processor request transfers, called NPR-level data transfers, are usually made for direct memory access (memory to/from mass storage).

Bus Requests

Bus requests from external devices can be made on one of five request lines. Highest priority is assigned to non-processor requests (NPR). These are direct memory access type transfers, and are honored by the processor during an instruction execution.

The processor's priority can be set under program control to one of eight levels using bits 7, 6, and 5 in the Processor Status Register. These bits set a priority level that inhibits granting of bus requests on lower levels or on the same level. When the processor's priority is set to a level, for example priority level 6, all bus requests on BR6 and below are ignored.

When more than one device is connected to the same bus request (BR) line, a device nearer the central processor has a higher priority than a device farther away. Any number of devices can be connected to a given BR or NPR line.

Thus the priority system is two-dimensional and provides each device with a unique priority. Each device may be dynamically enabled or disabled under program control.

Once a device other than the processor has control of the bus, it may do one of two types of operations: data transfers or interrupt operations.

NPR Data Transfers

NPR data transfers can be made between any two peripheral devices without the supervision of the processor. Normally, NPR transfers are between a mass storage device, such as a disk, and core memory. The structure of the bus also permits device-to-device transfers, allowing customer-designed peripheral controllers to access other devices, such as disks, directly.

An NPR device has very fast access to the bus and can transfer at high data rates once it has control. The processor state is not affected by the transfer; therefore the processor can relinquish control while an instruction is in progress. This can occur at the end of any bus cycles except in between a read-modify-write sequence. An NPR device in control of the bus may transfer 16-bit words from memory at memory speed.

BR Transfers

Devices that gain bus control with one of the bus request lines (BR7-BR4) can take full advantage of the central processor by requesting an interrupt. In this way, the entire instruction set is available for manipulating data and status registers.

When a service routine is to be run, the current task being performed by the central processor is interrupted, and the device service routine is initiated. Once the request has been satisfied, the processor returns to its former task.

Interrupt Procedure

Interrupt handling is automatic in the PDP-11. No device polling is required to determine which service routine to execute. The operations required to service an interrupt are as follows:

1. Processor relinquishes control of the bus, priorities permitting.
2. When a master gains control, it sends the processor an interrupt command and a unique memory address which contains the address of the device's service routine, called the interrupt vector address. Immediately following this pointer address is a word (located at vector address +2) which is to be used as a new Processor Status Word.
3. The processor stores the current Processor Status (PS) and the current Program Counter (PC) in CPU temporary registers.
4. The new PC and PS (interrupt vector) are taken from the specified address. The old PS and PC are then pushed onto the current stack. The service routine is then initiated.
5. The device service routine can cause the processor to resume the interrupted process by executing the Return from Interrupt instruc-

tion, described in Chapter 4, which pops the two top words from the current processor stack and uses them to load the PC and PS registers.

A device routine can be interrupted by a higher priority bus request any time after the new PC and PS have been loaded. If such an interrupt occurs, the PC and PS of the service routine are automatically stored in the temporary registers and then pushed onto the new current stack, and the new device routine is initiated.

Interrupt Servicing
Every hardware device capable of interrupting the processor has a unique set of locations (2 words) reserved for its interrupt vector. The first word contains the location of the device's service routine, and the second, the Processor Status Word that is to be used by the service routine. Through proper use of the PS, the programmer can switch the priority level of the processor, and modify the processor's priority level to mask out lower level interrupts.

Reentrant Code
Both the interrupt handling hardware and the subroutine call hardware facilitate writing reentrant code for the PDP-11. This type of code allows a single copy of a given subroutine or program to be shared by more than one process or task. This reduces the amount of core needed for multi-task applications such as the concurrent servicing of many peripheral devices.

Power Fail and Restart
Whenever ac power drops below 95 volts for 115V power (190 volts for 230V), the power fail sequence is initiated. The central processor automatically traps to location 24 and the power fail program has 2 msec to save all volatile information (data in registers), and to condition peripherals for power fail.

When power is restored the processor traps to location 24 and executes the power-up routine to restore the machine to its state prior to power failure. If MOS memory is present, and the battery backup system is depleted, the power-up will bootstrap the machine.

Note that the above automatic operation occurs under panel lock mode. In non-panel lock mode, the power-up will not bootstrap the machine. The power-up sequence can also be altered by console intervention.

CHAPTER 3

ADDRESSING MODES

Data stored in memory must be accessed, and manipulated. Data handling is specified by a PDP-11 instruction (MOV, ADD etc.) which usually indicates:

the function (operation code)

a general purpose register to be used when locating the source operand and/or a general purpose register to be used when locating the destination operand.

an addressing mode (to specify how the selected register(s) is/are to be used)

Since a large portion of the data handled by a computer is usually structured (in character strings, in arrays, in lists etc.), the PDP-11 has been designed to handle structured data efficiently and flexibly. The general registers may be used with an instruction in any of the following ways:

as accumulators. The data to be manipulated resides within the register.

as pointers. The contents of the register are the address of the operand, rather than the operand itself.

as pointers which automatically step through core locations. Automatically stepping forward through consecutive core locations is known as autoincrement addressing; automatically stepping backwards is known as autodecrement addressing. These modes are particularly useful for processing tabular data.

as index registers. In this instance the contents of the register, and the word following the instruction are summed to produce the address of the operand. This allows easy access to variable entries in a list.

PDP-11's also have instruction addressing mode combinations which facilitate temporary data storage structures for convenient handling of data which must be frequently accessed. This is known as the " stack."

In the PDP-11 any register can be used as a "stack pointer"under program control, however, certain instructions associated with subroutine linkage and interrupt service automatically use Register 6 as a "hardware stack pointer". For this reason R6 is frequently referred to as the "SP".

R7 is used by the processor as its program counter (PC). It is recommended that R7 not be used as a stack pointer.

An important PDP-11 feature, which must be considered in conjunction with the addressing modes, is the register arrangement;

Six general purpose registers, (R0-R5)

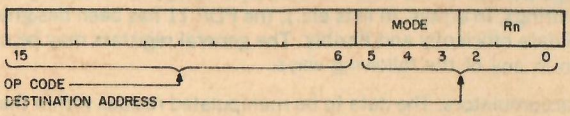
A hardware Stack Pointer (SP), register (R6)

A Program Counter (PC), register (R7).

Instruction mnemonics and address mode symbols are sufficient for writing machine language programs. The programmer need not be concerned about conversion to binary digits; this is accomplished automatically by the PDP-11 MACRO Assembler.

3.1 SINGLE OPERAND ADDRESSING

The instruction format for all single operand instructions (such as clear, increment, test) is:



Bits 15 through 6 specify the operation code that defines the type of instruction to be executed.

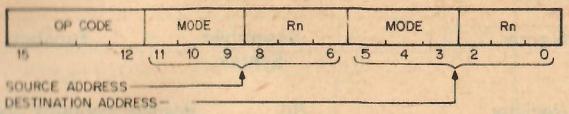
Bits 5 through 0 form a six-bit field called the destination address field. This consists of two subfields:

a) Bits 0 through 2 specify which of the eight general purpose registers is to be referenced by this instruction word.

b) Bits 3 through 5 specify how the selected register will be used (address mode). Bit 3 is set to indicate deferred (indirect) addressing.

3.2 DOUBLE OPERAND ADDRESSING

Operations which imply two operands (such as add, subtract, move and compare) are handled by instructions that specify two addresses. The first operand is called the source operand, the second the destination operand. Bit assignments in the source and destination address fields may specify different modes and different registers. The instruction format for the double operand instruction is:



The source address field is used to select the source operand, the first operand. The destination is used similarly, and locates the second operand and the result. For example, the instruction ADD A, B adds the contents (source operand) of location A to the contents (destination operand) of location B. After execution B will contain the result of the addition and the contents of A will be unchanged.

Examples in this section and further in this chapter use the following sample PDP-11 instructions:

Mnemonic	Description	Octal Code
CLR	clear (zero the specified destination)	0050DD
CLRB	clear byte (zero the byte in the specified destination)	1050DD
INC	increment (add 1 to contents of destination)	0052DD
INCB	increment byte (add 1 to the contents of destination byte)	1052DD
COM	complement (replace the contents of the destination by their logical complement; each 0 bit is set and each 1 bit is cleared)	0051DD
COMB	complement byte (replace the contents of the destination byte by their logical complement; each 0 bit is set and each 1 bit is cleared).	1051DD
ADD	add (add source operand to destination operand and store the result at destination address)	06SSDD

DD = destination field (6 bits)

SS = source field (6 bits)

( ) = contents of

3.3 DIRECT ADDRESSING

The following table summarizes the four basic modes used with direct addressing.

DIRECT MODES

Mode	Name	Assembler Syntax	Function
0	Register	Rn	Register contains operand
2	Autoincrement	(Rn)+	Register is used as a pointer to sequential data then incremented
4	Autodecrement	-(Rn)	Register is decremented and then used as a pointer.
6	Index	X(Rn)	Value X is added to (Rn) to produce address of operand. Neither X nor (Rn) are modified.

3.3.1 Register Mode

OPR Rn

With register mode any of the general registers may be used as simple accumulators and the operand is contained in the selected register. Since they are hardware registers, within the processor, the general registers operate at high speeds and provide speed advantages when used for operating on frequently-accessed variables. The PDP-11 assembler interprets and assembles instructions of the form OPR Rn as register mode operations. Rn represents a general register name or number and OPR is used to represent a general instruction mnemonic. Assembler syntax requires that a general register be defined as follows:

R0 = %0 (% sign indicates register definition)

R1 = %1

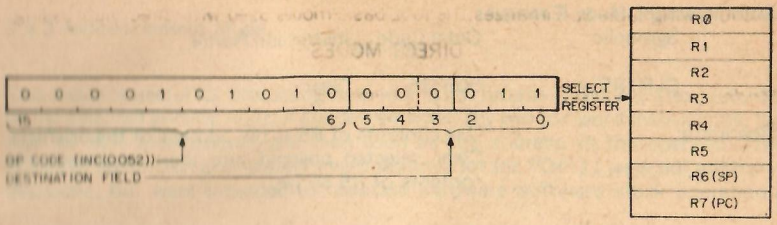
R2 = %2, etc.

Registers are typically referred to by name as R0, R1, R2, R3, R4, R5, R6 and R7. However R6 and R7 are also referred to as SP and PC, respectively.

Register Mode Examples
(all numbers in octal)

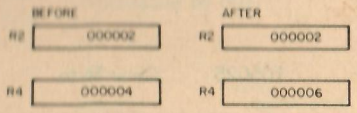
	Symbolic	Octal Code	Instruction Name
1.	INC R3	005203	Increment

Operation: Add one to the contents of general register 3



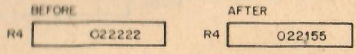
2. ADD R2,R4 060204 Add

Operation: Add the contents of R2 to the contents of R4.



3. COMB R4 105104 Complement Byte

Operation: One's complement bits 0-7 (byte) in R4. (When general registers are used, byte instructions only operate on bits 0-7; i.e. byte 0 of the register)



3.3.2 Autoincrement Mode

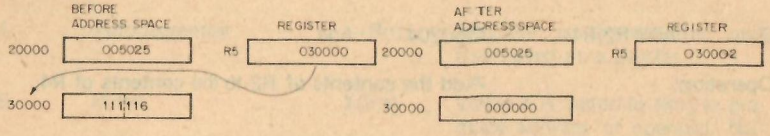
OPR (Rn)+

This mode provides for automatic stepping of a pointer through sequential elements of a table of operands. It assumes the contents of the selected general register to be the address of the operand. Contents of registers are stepped (by one for bytes, by two for words, always by two for R6 and R7) to address the next sequential location. The autoincrement mode is especially useful for array processing and stacks. It will access an element of a table and then step the pointer to address the next operand in the table. Although most useful for table handling, this mode is completely general and may be used for a variety of purposes.

Autoincrement Mode Examples

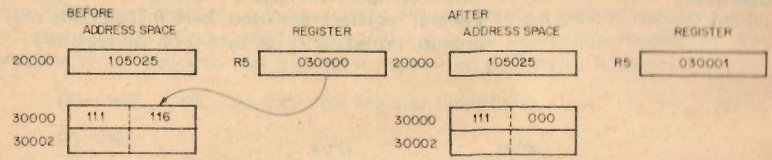
	Symbolic	Octal Code	Instruction Name
1.	CLR (R5)+	005025	Clear

Operation: Use contents of R5 as the address of the operand. Clear selected operand and then increment the contents of R5 by two.



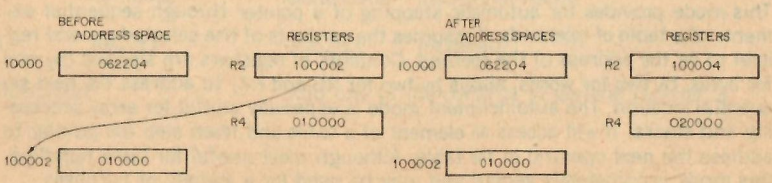
2.	CLRB (R5)+	105025	Clear Byte

Operation: Use contents of R5 as the address of the operand. Clear selected byte operand and then increment the contents of R5 by one.



3.	ADD (R2)+,R4	062204	Add

Operation: The contents of R2 are used as the address of the operand which is added to the contents of R4. R2 is then incremented by two.



3.3.3 Autodecrement Mode

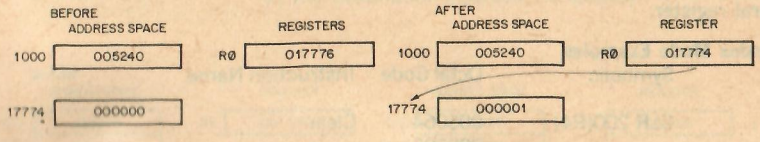
OPR–(Rn)

This mode is useful for processing data in a list in reverse direction. The contents of the selected general register are decremented (by two for word instructions, by one for byte instructions) and then used as the address of the operand. The choice of postincrement, predecrement features for the PDP-11 were not arbitrary decisions, but were intended to facilitate hardware/software stack operations.

Autodecrement Mode Examples

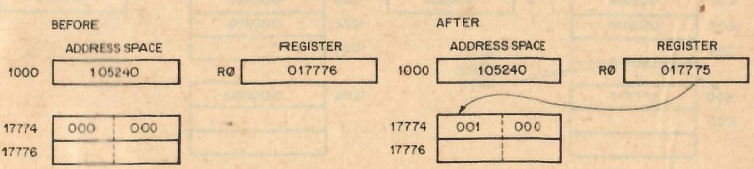
	Symbolic	Octal Code	Instruction Name
1.	INC–(R0)	005240	Increment

Operation: The contents of R0 are decremented by two and used as the address of the operand. The operand is increased by one.



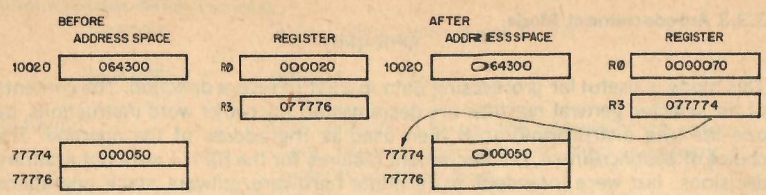
2.	INCB–(R0)	105240	Increment Byte

Operation: The contents of R0 are decremented by one then used as the address of the operand. The operand byte is increased by one.



3.	ADD–(R3),R0	064300	Add

Operation: The contents of R3 are decremented by 2 then used as a pointer to an operand (source) which is added to the contents of R0 (destination operand).



3.3.4 Index Mode

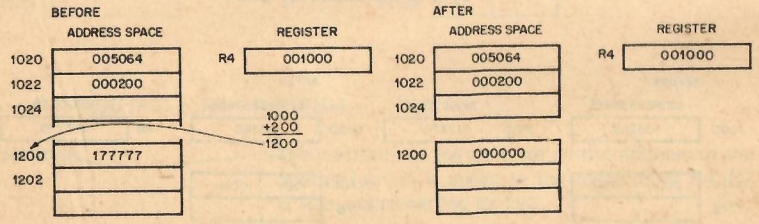
OPR X(Rn)

The contents of the selected general register, and an index word following the instruction word, are summed to form the address of the operand. The contents of the selected register may be used as a base for calculating a series of addresses, thus allowing random access to elements of data structures. The selected register can then be modified by program to access data in the table. Index addressing instructions are of the form OPR X(Rn) where X is the indexed word and is located in the memory location following the instruction word and Rn is the selected general register.

Index Mode Examples

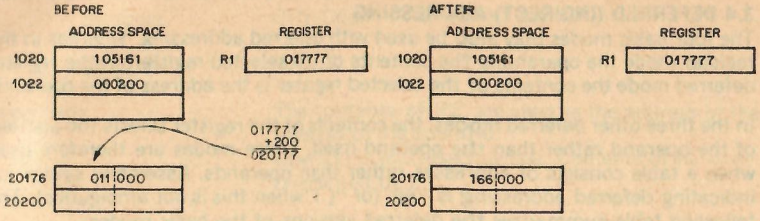
	Symbolic	Octal Code	Instruction Name
1.	CLR 200(R4)	005064 000200	Clear

Operation: The address of the operand is determined by adding 200 to the contents of R4. The location is then cleared.



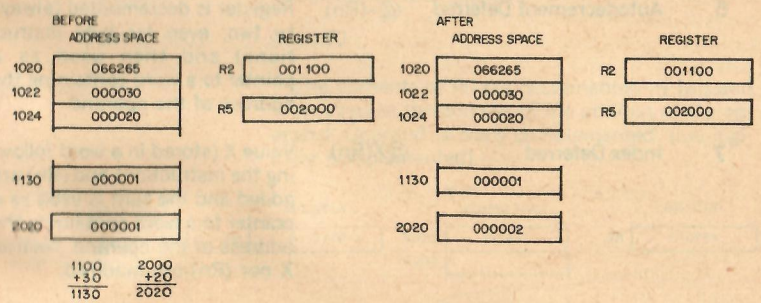
2.	COMB 200(R1)	105161 000200	Complement Byte

Operation: The contents of a location which is determined by adding 200 to the contents of R1 are one's complemented. (i.e. logically complemented)



3. ADD 30(R2),20(R5) 066265 Add
000030
000020

Operation: The contents of a location which is determined by adding 30 to the contents of R2 are added to the contents of a location which is determined by adding 20 to the contents of R5. The result is stored at the destination address, i.e. 20(R5)



3.4 DEFERRED (INDIRECT) ADDRESSING

The four basic modes may also be used with deferred addressing. Whereas in the register mode the operand is the contents of the selected register, in the register deferred mode the contents of the selected register is the address of the operand.

In the three other deferred modes, the contents of the register selects the address of the operand rather than the operand itself. These modes are therefore used when a table consists of addresses rather than operands. Assembler syntax for indicating deferred addressing is "@" (or "( )" when this is not ambiguous). The following table summarizes the deferred versions of the basic modes:

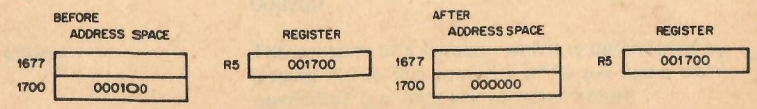
Mode	Name	Assembler Syntax	Function
1	Register Deferred	@Rn or (Rn)	Register contains the address of the operand
3	Autoincrement Deferred	@(Rn)+	Register is first used as a pointer to a word containing the address of the operand, then incremented (always by 2; even for byte instructions).
5	Autodecrement Deferred	@–(Rn)	Register is decremented (always by two; even for byte instructions) and then used as a pointer to a word containing the address of the operand
7	Index Deferred	@X(Rn)	Value X (stored in a word following the instruction) and (Rn) are added and the sum is used as a pointer to a word containing the address of the operand. Neither X nor (Rn) are modified.

Since each deferred mode is similar to its basic mode counterpart, separate descriptions of each deferred mode are not necessary. However, the following examples illustrate the deferred modes.

Register Deferred Mode Example

Symbolic	Octal Code	Instruction Name
CLR @R5	005015	Clear

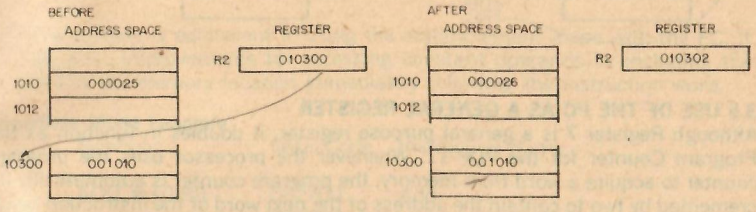
Operation: The contents of location specified in R5 are cleared.



Autoincrement Deferred Mode Example

Symbolic	Octal Code	Instruction Name
INC@(R2)+	005232	Increment

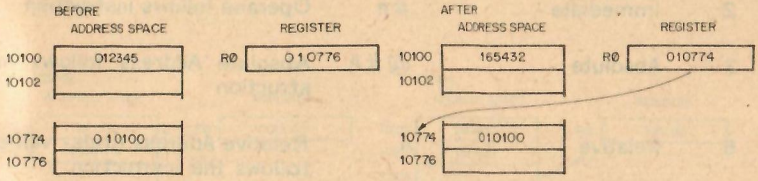
Operation: The contents of R2 are used as the address of the address of the operand.
Operand is increased by one. Contents of R2 is incremented by 2.



Autodecrement Deferred Mode Example

Symbolic	Octal Code	Complement
COM @-(R0)	005150	

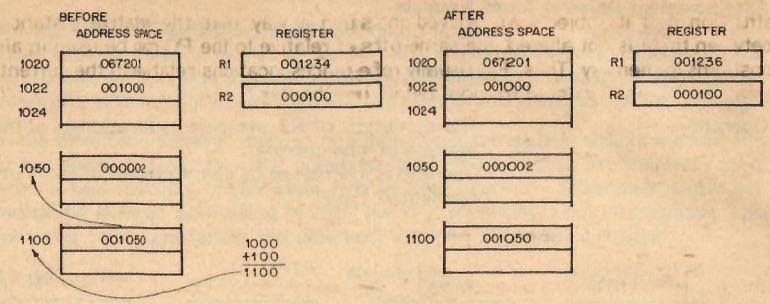
Operation: The contents of R0 are decremented by two and then used as the address of the address of the operand. Operand is one's complemented. (i.e. logically complemented)



Index Deferred Mode Example

Symbolic	Octal Code	Instruction Name
ADD @ 1000(R2),R1	067201 001000	Add

Operation: 1000 and contents of R2 are summed to produce the address of the address of the source operand the contents of which are added to contents of R1; the result is stored in R1.



3.5 USE OF THE PC AS A GENERAL REGISTER

Although Register 7 is a general purpose register, it doubles in function as the Program Counter for the PDP-11. Whenever the processor uses the program counter to acquire a word from memory, the program counter is automatically incremented by two to contain the address of the next word of the instruction being executed or the address of the next instruction to be executed. (When the program uses the PC to locate byte data, the PC is still incremented by two.)

The PC responds to all the standard PDP-11 addressing modes. However, there are four of these modes with which the PC can provide advantages for handling position independent code (PIC - seeChapter 5) and unstructured data. When regarding the PC these modes are termed immediate, absolute (or immediate deferred), relative and relative deferred, and are summarized below:

Mode	Name	Assembler Syntax	Function
2	Immediate	#n	Operand follows instruction
3	Absolute	@#A	Absolute Address follows instruction
6	Relative	A	Relative Address (index value) follows the instruction.
7	Relative Deferred	@A	Index value (stored in the word following the instruction) is the relative address for the address of the operand.

The reader should remember that the special effect modes are the same as modes described in 3.3 and 3.4, but the general register selected is R7, the program counter.

When a standard program is available for different users, it often is helpful to be able to load it into different areas of core and run it there. PDP-11's can accomplish the relocation of a program very efficiently through the use of position inde-

pendent code (PIC) which is written by using the PC addressing modes. If an instruction and its objects are moved in such a way that the relative distance between them is not altered, the same offset relative to the PC can be used in all positions in memory. Thus, PIC usually references locations relative to the current location. PIC is discussed in more detail in Chapter 5.

The PC also greatly facilitates the handling of unstructured data. This is particularly true of the immediate and relative modes.

3.5.1 Immediate Mode

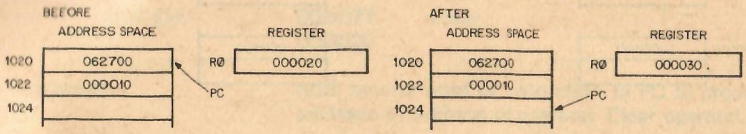
OPR #n,DD

Immediate mode is equivalent to using the autoincrement mode with the PC. It provides time improvements for accessing constant operands by including the constant in the memory location immediately following the instruction word.

Immediate Mode Example

Symbolic	Octal Code	Instruction Name
ADD #10,R0	062700 000010	Add

Operation: The value 10 is located in the second word of the instruction and is added to the contents of R0. Just before this instruction is fetched and executed, the PC points to the first word of the instruction. The processor fetches the first word and increments the PC by two. The source operand mode is 27 (autoincrement the PC). Thus, the PC is used as a pointer to fetch the operand (the second word of the instruction) before being incremented by two to point to the next instruction.



3.5.2 Absolute Addressing

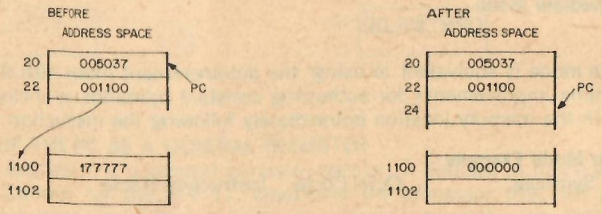
OPR @#A

This mode is the equivalent of immediate deferred or autoincrement deferred using the PC. The contents of the location following the instruction are taken as the address of the operand. Immediate data is interpreted as an absolute address (i.e., an address that remains constant no matter where in memory the assembled instruction is executed).

Absolute Mode Examples

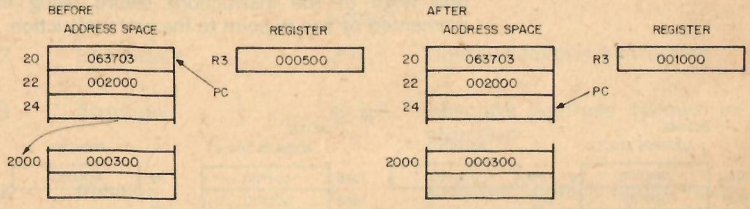
	Symbolic	Octal Code	Instruction Name
1.	CLR @#1100	005037 001100	Clear

Operation: Clear the contents of location 1100.



2.	ADD @#2000,R3	063703 002000	

Operation: Add contents of location 2000 to R3.



3.5.3 Relative Addressing

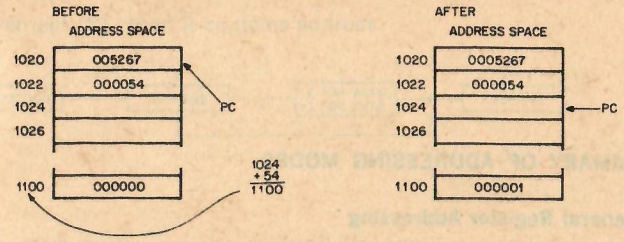
OPR A or OPR X(PC)
where X is the location of A relative to the instruction.

This mode is assembled as index mode using R7. The base of the address calculation, which is stored in the second or third word of the instruction, is not the address of the operand, but the number which, when added to the (PC), becomes the address of the operand. This mode is useful for writing position independent code (see Chapter 5) since the location referenced is always fixed relative to the PC. When instructions are to be relocated, the operand is moved by the same amount.

Relative Addressing Example

Symbolic	Octal Code	Instruction Name
INC A	005267 000054	Increment

Operation: To increment location A, contents of memory location immediately following instruction word are added to (PC) to produce address A. Contents of A are increased by one.



3.5.4 Relative Deferred Addressing

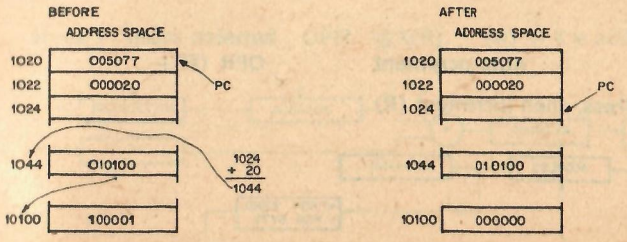
OPR@A or OPR@X(PC), where x is location containing address of A, relative to the instruction.

This mode is similar to the relative mode, except that the second word of the instruction, when added to the PC, contains the address of the address of the operand, rather than the address of the operand.

Relative Deferred Mode Example

Symbolic	Octal Code	Instruction Name
CLR @A	005077 000020	Clear

Operation: Add second word of instruction to PC to produce address of address of operand. Clear operand.



3.6 USE OF STACK POINTER AS GENERAL REGISTER

The processor stack pointer (SP, Register 6) is in most cases the general register used for the stack operations related to program nesting. Autodecrement with Register 6 "pushes" data on to the stack and autoincrement with Register 6 "pops" data off the stack. Index mode with SP permits random access of items on the stack. Since the SP is used by the processor for interrupt handling, it has a special attribute: autoincrements and autodecrements are always done in steps of two. Byte operations using the SP in this way leave odd addresses unmodified.

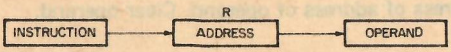
3.7 SUMMARY OF ADDRESSING MODES

3.7.1 General Register Addressing

R is a general register, 0 to 7
(R) is the contents of that register

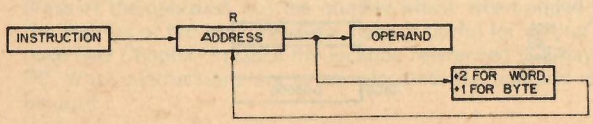
Mode 0 **Register** OPR R R contains operand

Mode 1 **Register deferred** OPR (R) R contains address

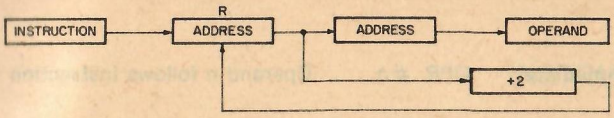


Mode 2 **Auto-increment** OPR (R)+

R contains address, then increment (R)

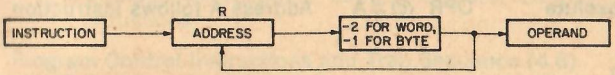


Mode 3 **Auto-increment deferred** OPR @(R)+ R contains address of address, then increment (R) by 2

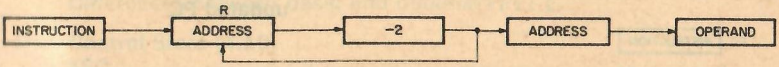


Mode 4 **Auto-decrement** OPR —(R)

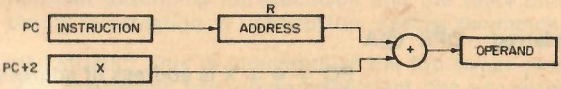
Decrement (R), then R contains address



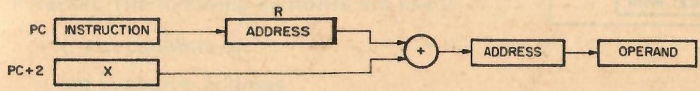
Mode 5 **Auto-decrement deferred** OPR @—(R) Decrement (R) by 2, then R contains address of address



Mode 6 **Index** OPR X(R) (R) + X is address



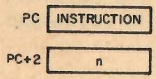
Mode 7 **Index deferred** OPR @X(R) (R) + X is address of address



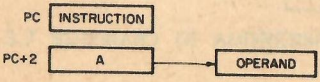
3.7.2 Program Counter Addressing

Register = 7

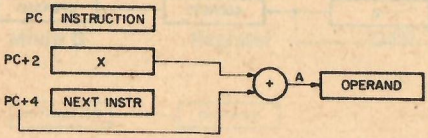
Mode 2 **Immediate** OPR #n Operand n follows instruction



Mode 3 **Absolute** OPR @#A Address A follows instruction

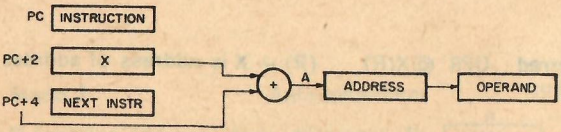


Mode 6 **Relative** OPR A $\underbrace{PC + 4}_{\text{updated PC}} + X$ is address



Mode 7 **Relative deferred** OPR @A

$\underbrace{PC + 4}_{\text{updated PC}} + X$ is address of address



CHAPTER 4

INSTRUCTION SET

4.1 INTRODUCTION

This chapter describes the PDP-11/60 instructions in the following order:

Single Operand (4.4)
General Shifts, Multiple Precision, Rotates

Double Operand (4.5)
Arithmetic Instructions, General Register Destination, Logical Instructions

Program Control Instructions and Trap Sequence (4.6)
Branches, Subroutines, Traps: Instructions, Reserved, Illegal, Overflow (Stack Limit), Bus Error, Trace

Miscellaneous (4.7)
Halt, Wait, Memory Management

Condition Code Operators (4.8)

Floating Point Instruction Set (4.9)
Differences between basic and optional FP11-E

User Control Store (4.10)
XFC

Maintenance (4.11)
MED

The specification for each instruction includes the mnemonic, octal code, binary code, a diagram showing the format of the instruction, a symbolic notation describing its execution and the effect on the condition codes, timing information, a description, special comments, and examples.

MNEMONIC: This is indicated at the top corner of each page. When the word instruction has a byte equivalent, the byte mnemonic is also shown.

INSTRUCTION FORMAT: A diagram accompanying each instruction shows the octal op code, the binary op code, and bit assignments. (Note that in byte instructions the most significant bit (bit 15) is always a 1.)

OPERATION: The operation of each instruction is described with a single notation. The following symbols are used:

( ) = contents of

src = source address

dst = destination address

loc = location

← = becomes

↑ = "is popped from stack"

↓ = "is pushed onto stack"

Λ = boolean AND

v = boolean OR

⊻ = exclusive OR

~ = boolean not

Reg or R = register

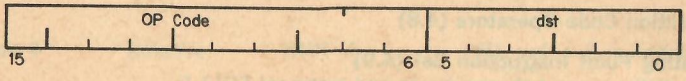
B = Byte

Appendix C contains an instruction set index which provides an alphabetic listing of the instructions.

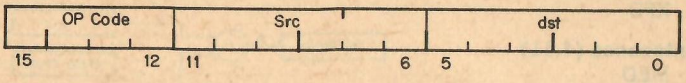
4.2 INSTRUCTION FORMATS

The major instruction formats are:

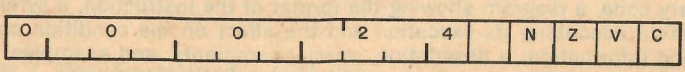
Single Operand Group



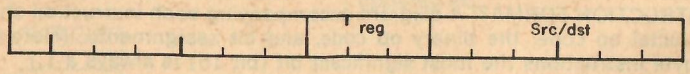
Double Operand Group



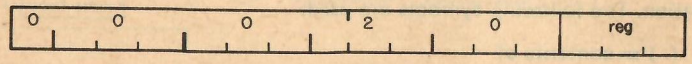
Condition Code Operators



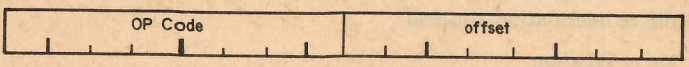
Register-Source or Destination



Subroutine Return

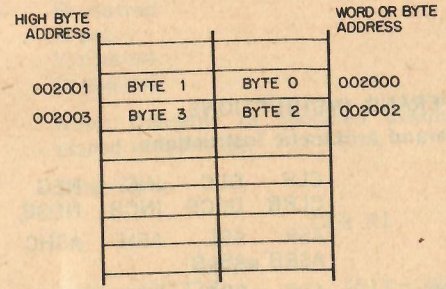


Branch



4.3 BYTE INSTRUCTIONS

The PDP-11 processors include a full complement of instructions that manipulate byte operands. Since all PDP-11 addressing is byte-oriented, byte manipulation addressing is straightforward. Byte instructions with autoincrement or autodecrement direct addressing cause the specified register to be modified by one to point to the next byte of data. Byte operations in register mode access the low-order byte of the specified register. These provisions enable the PDP-11 to perform as either a word or byte processor. The numbering scheme for word and byte addresses in core memory is:



The most significant bit (Bit 15) of the instruction word is set to indicate a byte instruction.

Example:

Symbolic	Octal	
CLR	0050DD	clear word
CLRB	1050DD	clear byte

4.4 SINGLE OPERAND INSTRUCTIONS

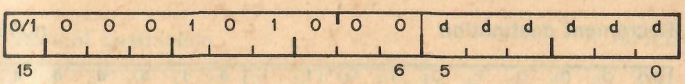
4.4.1 Single Operand Arithmetic Instructions

General:	CLR	DEC	INC	NEG	TST	COM
	CLRB	DECB	INCB	NEGB	TSTB	COMB
Shifts:	ASR	ASL	ASH	ASHC		
	ASRB	ASLB				
Multiple Precision:	ADC	SBC	SXT			
	ADCB	SBCB				
Rotates:	ROL	ROR	SWAB			
	ROLB	RORB				
Floating:	FADD	FSUB		FMUL		FDIV
	(not used in PDP-11/60)					

CLR
CLRB

Clear destination

n050DD



temp ←(dst)

Operation: (dst)←0

Condition Codes:
N: cleared
Z: set
V: cleared
C: cleared

Description: Word: Contents of specified destination are replaced with zeroes.

Byte: Same

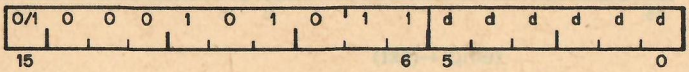
Example:

CLR R1

Before	After
(R1) = 177777	(R1) = 000000
N Z V C	N Z V C
1 1 1 1	0 1 0 0

DEC
DECB

Decrement destination n053DD



Operation: (dst)←(dst)−1

Condition Codes:
N: set if result is <0; cleared otherwise
Z: set if result is 0; cleared otherwise
V: set if (dst) was 100000; cleared otherwise
C: not affected

Description:
Word: Subtract 1 from the contents of the destination
Byte: Same

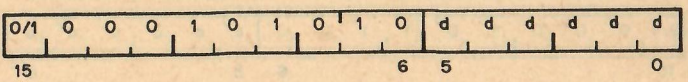
Example:

DEC R5

Before	After
(R5) = 000001	(R5) = 000000
N Z V C	N Z V C
1 0 0 0	0 1 0 0

INC
INCB

Increment destination n052DD



Operation: (dst)←(dst) + 1

Condition Codes:
N: set if result is <0; cleared otherwise
Z: set if result is 0; cleared otherwise
V: set if (dst) Was 077777; cleared otherwise
C: not affected

Description: Word: Add one to contents of destination
Byte: Same

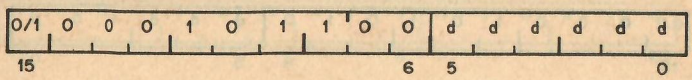
Example:

INC R2

Before	After
(R2) = 000333	(R2) = 000334
N Z V C	N Z V C
0 0 0 0	0 0 0 0

NEG
NEGB

Negate destination n0054DD



Operation: (dst) ← −(dst)

Condition Codes:
N: set if the result is <0; cleared otherwise
Z: set if result is 0; cleared otherwise
V: set if the result is 100000; cleared otherwise
C: cleared if the result is 0; set otherwise

Description: Word: Replaces the contents of the destination address by its two's complement. Note that 100000 is replaced by itself (in two's complement notation the most negative number has no positive counterpart).
Byte: Same

Example:

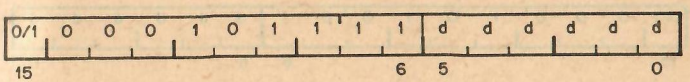
NEG R0

Before	After
(R0) = 000010	(R0) = 177770
N Z V C	N Z V C
0 0 0 0	1 0 0 1

TST
TSTB

Test destination

n057DD



Operation: (dst)←(dst)

Condition Codes:
N: set if the result is <0; cleared otherwise
Z: set if result is 0; cleared otherwise
V: cleared
C: cleared

Description: Word: Sets the condition codes N and Z according to the contents of the destination address
Byte: Same

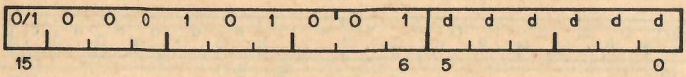
Example:

TST R1

Before	After
(R1) = 012340	(R1) = 012340
N Z V C	N Z V C
0 0 1 1	0 0 0 0

COM
COMB

Complement destination n051DD



Operation: (dst) ← ~ (dst)

Condition Codes:
N: set if most significant bit of result is set; cleared otherwise
Z: set if result is 0; cleared otherwise
V: cleared
C: set

Description: Replaces the contents of the destination address by their logical complement (each bit equal to 0 is set and each bit equal to 1 is cleared)
Byte: Same

Example:

COM R0

Before	After
(R0) = 013333	(R0) = 164444
N Z V C	N Z V C
0 1 1 0	1 0 0 1

4.4.2 Shifts

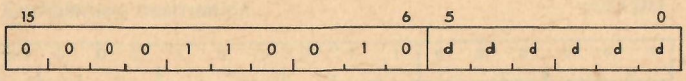
Scaling data by factors of two is accomplished by the shift instructions:

ASR—Arithmetic shift right	ASC—Multiple shift one word
ASL—Arithmetic shift left	ASC—Multiple shift one word

The sign bit (bit 15) of the operand is replicated in shifts to the right. The low order bit is filled with 0 in shifts to the left. Bits shifted out of the C bit, as shown in the following examples, are lost.

ASR

Arithmetic Shift Right destination 0062DD

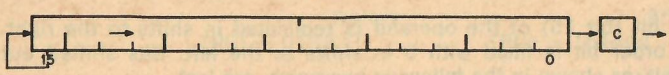


Operation: (dst) ←(dst) shifted one place to the right

Condition Codes:
N: set if the high-order bit of the result is set (result < 0); cleared otherwise
Z: set if the result = 0; cleared otherwise
V: loaded from the Exclusive OR of the N-bit and C-bit (as set by the completion of the shift operation)
C: loaded from low-order bit of the destination

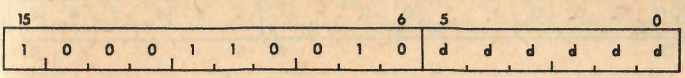
Description: Word: Shifts all bits of the destination right one place. Bit 15 is replicated. The C-bit is loaded from bit 0 of the destination. ASR performs signed division of the destination by two.

Word:



Arithmetic Shift Right destination

1062DD



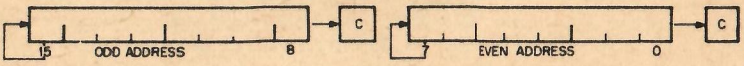
Operation: (dst)←(dst) shifted one place to the right

Condition Codes:
N: set if the high-order bit of the result is set (result < 0); cleared otherwise
Z: set if the result = 0; cleared otherwise
V: loaded from the Exclusive OR of the N-bit and C-bit (as set by the completion of the shift operation)
C: loaded from low-order bit of the destination

Description: Byte: Shift all bits of the destination right one place. Bit 15 or 7 is replicated, the C bit of status is loaded from bit 8 or 0, for odd or even address, respectively.

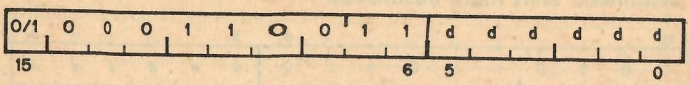
Note: In the PDP-11/60, the ASRB does a DATI/DATIP/DATO bus sequence in the execution portion of the instruction. This allows an interlocking of memory addresses. If an I/O page reference is made, the ASRB does a DATIP/DATIP/DATO bus sequence during instruction execution.

Byte:



ASL
ASLB

Arithmetic Shift Left destination n063DD

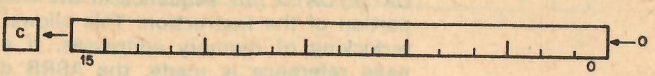


Operation: (dst)←(dst) shifted one place to the left

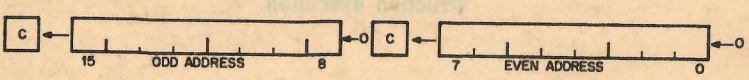
Condition Codes:
N: set if high-order bit of the result is set (result < 0); cleared otherwise
Z: set if the result = 0; cleared otherwise
V: loaded with the exclusive OR of the N-bit and C-bit (as set by the completion of the shift operation)
C: loaded with the high-order bit of the destination

Description: Word: Shifts all bits of the destination left one place. Bit 0 is loaded with an 0. The C-bit of the status word is loaded from the most significant bit of the destination. ASL performs a signed multiplication of the destination by 2 with overflow indication.

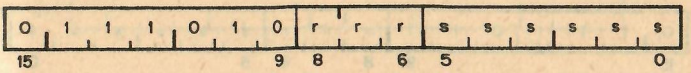
Word:



Byte:



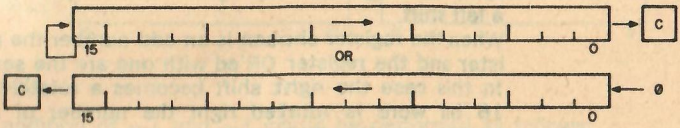
Shift Arithmetically 072RSS



Operation: R← R Shifted arithmetically NN places to right or left
Where NN = (src)

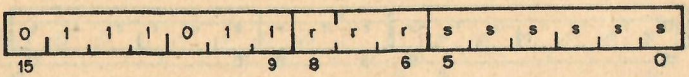
Condition Codes: N: set if result <0; cleared otherwise
Z: set if result = 0; cleared otherwise
V: set if sign of register changed during shift; cleared otherwise
C: loaded from last bit shifted out of register

Description: The contents of the register are shifted right or left the number of times specified by the source operand. The shift count is taken as the low order 6 bits of the source operand. This number ranges from −32 to +31. Negative is a right shift and positive is a left shift.



ASHC

Arithmetic Shift Combined 073RSS



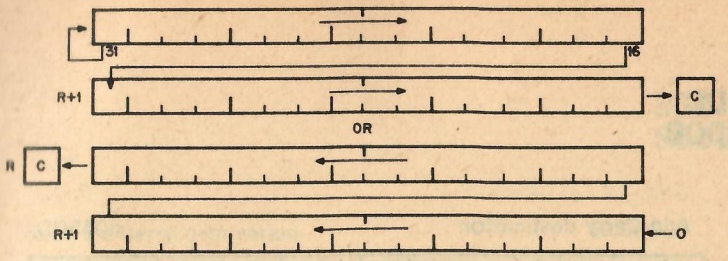
Operation: R, Rv1 ←R, Rv1 The double word is shifted NN places to the right or left, where NN = (src)

Condition Codes:
N: set if result <0; cleared otherwise
Z: set if result =0; cleared otherwise
V: set if sign bit changes during the shift; cleared otherwise
C: loaded with high order bit when left shift; loaded with low order bit when right shift (loaded with the last bit shifted out of the 32-bit operand)

Description: The contents of the register and the register ORed with one are treated as one 32 bit word, R + 1 (bits 0-15) and R (bits 16-31) are shifted right or left the number of times specified by the shift count. The shift count is taken as the low order 6 bits of the source operand. This number ranges from −32 to +31. Negative is a right shift and positive is a left shift.
When the register chosen is an odd number the register and the register OR'ed with one are the same. In this case the right shift becomes a rotate. The 16 bit word is rotated right the number of bits specified by the shift count.

When the register chosen is an odd number the register and the register OR'ed with one are the same. In this case the right shift becomes a rotate. The 16 bit word is rotated right the number of bits specified by the shift count provided the count is ≤ 16.

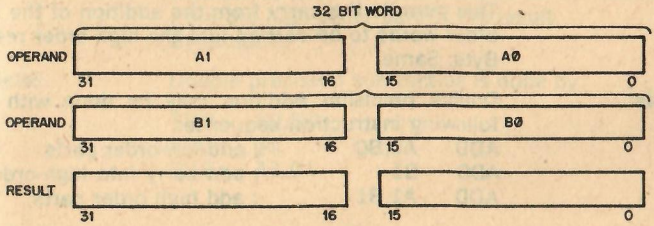
If an even register is specified, the sign of the 32 bit operand is bit 15 of the shifted result of the even register. If an odd register is specified, the high word is copied from the odd register and the resultant sign is bit 15 of the shifted result in the high word.



4.4.3 Multiple Precision

It is sometimes necessary to do arithmetic on operands considered as multiple words or bytes. The PDP-11 makes special provision for such operations with the instructions ADC (Add Carry) and SBC (Subtract Carry) and their byte equivalents and SXT (Sign Extend).

For example two 16-bit words may be combined into a 32-bit double precision word and added or subtracted as shown below:



Example:

The addition of −1 and −1 could be performed as follows:

−1 = 37777777777

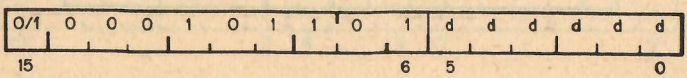
(R1) = 177777 (R2) = 177777 (R3) = 177777 (R4) = 177777

```
ADD   R1, R2
ADC   R3
ADD   R4,R3
```

1. After (R1) and (R2) are added, 1 is loaded into the C bit
2. ADC instruction adds C bit to (R3); (R3) = 0
3. (R3) and (R4) are added
4. Result is 37777777776 or −2

ADC
ADCB

Add Carry destination n055DD



Operation: (dst) ←(dst) + (C)

Condition Codes:
N: set if result <0; cleared otherwise
Z: set if result =0; cleared otherwise
V: set if (dst) was 077777 and (C) was 1: cleared otherwise
C: set if (dst) was 177777 and (C) was 1; cleared otherwise

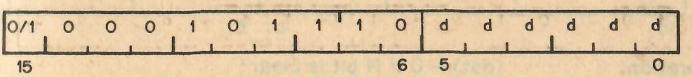
Description: Adds the contents of the C-bit into the destination. This permits the carry from the addition of the low-order words to be carried into the high-order result.
Byte: Same

Example: Double precision addition may be done with the following instruction sequence:

```
ADD   A0,B0      ; add low-order parts
ADC   B1         ; add carry into high-order
ADD   A1,B1      ; add high order parts
```

SBC
SBCB

Subtract Carry destination n056DD



Operation: (dst) ←(dst)—(C)

Condition Codes:
N: set if result <0; cleared otherwise
Z: set if result 0; cleared otherwise
V: set if(dst)was 100000; cleared otherwise
C: set if (dst) was 0 and C was 1; clearedotherwise

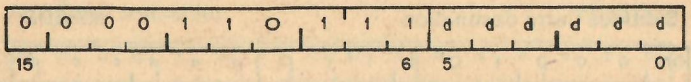
Description: Word: Subtracts the contents of the C-bit from the destination. This permits the carry from the subtraction of two low-order words to be subtracted from the high order part of the result.
Byte: Same

Example: Double precision subtraction is done by:

```
SUB   A0,B0
SBC   B1
SUB   A1,B1
```

SXT

Sign Extend destination 0067DD



Operation: (dst) ← 0 if N bit is clear
(dst) ← —1 N bit is set

Condition Codes: N: unaffected
Z: set if N bit clear
V: cleared
C: unaffected

Description: If the condition code bit N is set then a —1 is placed in the destination operand: if N bit is clear, then a 0 is placed in the destination operand. This instruction is particularly useful in multiple precision arithmetic because it permits the sign to be extended through multiple words.

Example:

SXT A

Before	After
(A) = 012345	(A) = 177777
N Z V C	N Z V C
1 0 0 0	1 0 0 0

4.4.4 Rotates

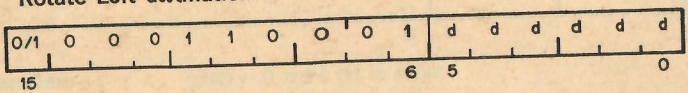
The rotate instructions operate on the destination word and the C bit as though they formed a 17-bit "circular buffer." These instructions facilitate sequential bit testing and detailed bit manipulation.

ROL
ROR
SWAB

ROL
ROLB

Rotate Left destination

n061DD



Condition Codes:

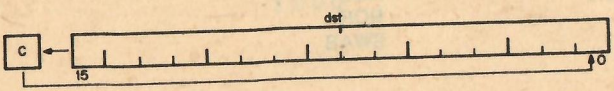
N: set if the high-order bit of the result word is set (result <0): cleared otherwise
Z: set if all bits of the result word = 0; cleared otherwise
V: loaded with the Exclusive OR of the N-bit and C-bit (as set by the completion of the rotate operation)
C: loaded with the high-order bit of the destination

Description:

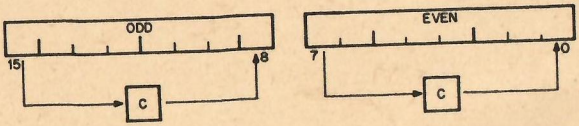
Word: Rotate all bits of the destination left one place. Bit 15 is loaded into the C-bit of the status word and the previous contents of the C-bit are loaded into Bit 0 of the destination.
Byte: Same

Example:

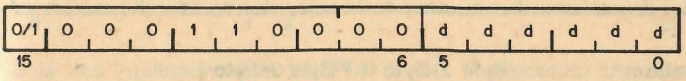
Word:



Bytes:



Rotate Right destination n060DD



Condition Codes:

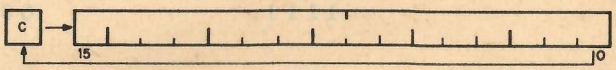
N: set if the high-order bit of the result is set (result $<$ 0); cleared otherwise
Z: set if all bits of result $=$ 0; cleared otherwise
V: loaded with the Exclusive OR of the N-bit and C-bit (as set by the completion of the rotate operation)
C: loaded with the low-order bit of the destination

Description:

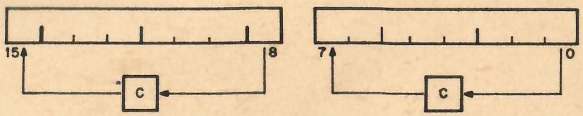
Rotates all bits of the destination right one place. Bit 0 is loaded into the C-bit and the previous contents of the C-bit are loaded into bit 15 of the destination.
Byte: Same

Example:

Word:

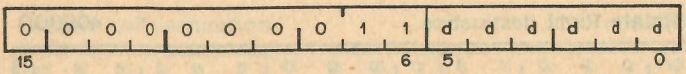


Byte:



SWAB

Swap Bytes destination 0003DD



Operation: Byte 1/Byte 0 ←Byte 0/Byte 1

Condition Codes:
N: set if high-order bit of low-order byte (bit 7) of result is set; cleared otherwise
Z: set if low-order byte of result = 0; cleared otherwise
V: cleared
C: cleared

Description: Exchanges high-order byte and low-order byte of the destination word (destination must be a word address).

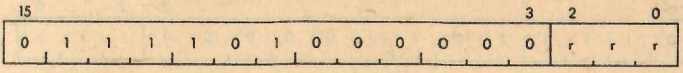
Example:

SWAB R1

Before	After
(R1) = 077777	(R1) = 177577
N Z V C	N Z V C
1 1 1 1	0 0 0 0

FADD

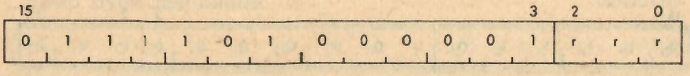
floating add 07500R



FADD is not implemented in the PDP-11/60. A processor trap through vector address 10 occurs for reserved instruction.

FSUB

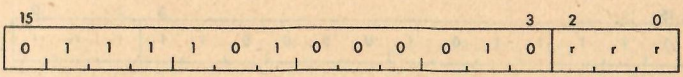
floating subtract 07501R



FSUB is not implemented in the PDP-11/60. A processor trap through vector address 10 occurs for reserved instruction.

FMUL

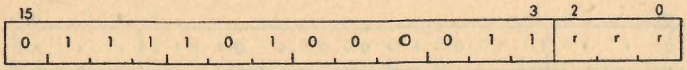
floating multiply 07502R



FMUL is not implemented in the PDP-11/60. A processor trap through vector address 10 occurs for reserved instruction.

FDIV

floating divide 07503R



FDIV is not implemented in the PDP-11/60. A processor trap through vector address 10 occurs for reserved instruction.

4.5 DOUBLE OPERAND INSTRUCTIONS

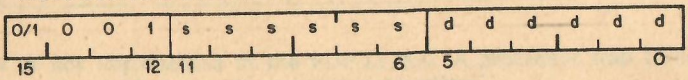
Double operand instructions provide an instruction (and time) saving facility since they eliminate the need for "load" and "save" sequences such as those used in accumulator-oriented machines.

General:	MOV MOVB	ADD	SUB	CMP CMPB
Register Destination:	MUL	DIV	XOR	
Logical:	BIS BISB	BIT BITB	BIC BICB	

4.5.1 Double Operand General Instructions

MOV
MOVB

Move source to destination — n1SSDD



Operation: (dst) ←(src)

Condition Codes:
N: set if (src) <0; cleared otherwise
Z: set if (src) = 0; cleared otherwise
V: cleared
C: not affected

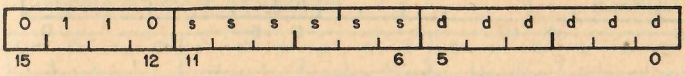
Description: Word: Moves the source operand to the destination location. The previous contents of the destination are lost. The contents of the source address are not affected.
Byte: Same as MOV. The MOVB to a register (unique among byte instructions) extends the most significant bit of the low order byte (sign extension). Otherwise MOVB operates on bytes exactly as MOV operates on words.

Example:

```
MOV   XXX,R1          ; loads Register 1 with the contents of memory location; XXX represents a programmer-defined mnemonic used to represent a memory location

MOV   #20,R0          ; loads the number 20 into Register 0; "#" indicates that the value 20 is the operand

MOV   @#20,-(R6)      ; pushes the operand contained in location 20 onto the stack

MOV   (R6)+,@#177566  ; pops the operand off a stack and moves it into memory location 177566 (terminal print buffer)

MOV   R1,R3           ; performs an interregister transfer

MOVB  @#177562,@#177566  ; moves a character from terminal keyboard buffer to terminal buffer
```

ADD

Add source to destination 06SSDD



Operation: (dst) ←(src) + (dst)

Condition Codes:
N: set if result <0; cleared otherwise
Z: set if result = 0; cleared otherwise
V: set if there was arithmetic overflow as a result of the operation; that is both operands were of the same sign and the result was of the opposite sign; cleared otherwise
C: set if there was a carry from the most significant bit of the result; cleared otherwise

Description: Adds the source operand to the destination operand and stores the result at the destination address. The original contents of the destination are lost. The contents of the source are not affected. Two's complement addition is performed.

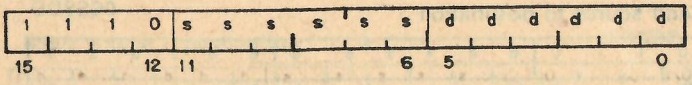
Examples:

Add to register:	ADD	20,R0
Add to memory:	ADD	R1,XXX
Add register to register:	ADD	R1,R2
Add memory to memory:	ADD	@#17750,XXX

XXX is a programmer-defined mnemonic for a memory location.

SUB

Subtract source from destination 16SSDD



Operation: (dst) ← (dst) — (src)

Condition Codes:
N: set if result <0; cleared otherwise
Z: set if result =0; cleared otherwise
V: set if there was arithmetic overflow as a result of the operation, that is if operands were of opposite signs and the sign of the source was the same as the sign of the result; cleared otherwise
C: cleared if there was a carry from the most significant bit of the result; set otherwise

Description: Subtracts the source operand from the destination operand and leaves the result at the destination address. The original contents of the destination are lost. The contents of the source are not affected. In double-precision arithmetic the C-bit, when set, indicates a "borrow"

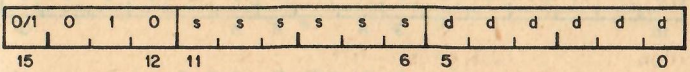
Example:

SUB R1, R2

Before	After
(R1) = 011111	(R1) = 011111
(R2) = 012345	(R2) = 001234
N Z V C	N Z V C
1 1 1 1	0 0 0 0

CMP
CMPB

Compare source to destination n2SSDD



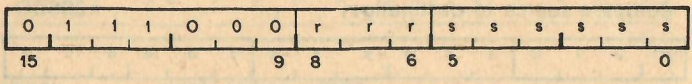
Operation: (src)—(dst) [in detail, (src) + ~ (dst) + 1]

Condition Codes:
N: set if result <0; cleared otherwise
Z: set if result =0; cleared otherwise
V: set if there was arithmetic overflow; that is, operands were of opposite signs and the sign of the destination was the same as the sign of the result; cleared otherwise
C: cleared if there was a carry from the most significant bit of the result; set otherwise

Description:
Compares the source and destination operands and sets the condition codes, which may then be used for arithmetic and logical conditional branches. Both operands are unaffected. The only action is to set the condition codes. The compare is customarily followed by a conditional branch instruction.
Note that unlike the subtract instruction the order of operation is (src)—(dst), not (dst)—(src).

MUL

Multiply 070RSS



Operation: R, Rv| ← R x(src)

Conditon Codes:
N: set if product is <0; cleared otherwise
Z: set if product is 0; cleared otherwise
V: cleared
C: set if the result is less than -2^{15} or greater than or equal to $2^{15}-1$.

Description: The contents of the destination register and source taken as two's complement integers are multiplied and stored in the destination register and the succeeding register (if R is even). If R is odd only the low order product is stored. Assembler syntax is: MUL S,R.

If an even register is specified, the 32-bit product is stored with the high 16-bits in the even register and the low 16-bits in the odd register.

If an odd register is specified for a multiply, only the low-order 16-bits of the product is stored. The condition codes are set according to the internally generated 32-bit product.

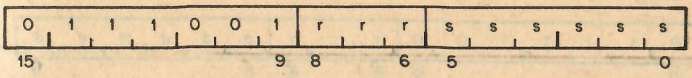
(Note that the actual destination is R,Rv1 which reduces to just R when R is odd.)

Example: 16-bit product (R is odd)

```
CLC                 ;Clear carry condition code
MOV #400,R1
MUL #10,R1
BCS ERROR           ;Carry will be set if
                    ;product is less than
                    ;-2^15 or greater than or equal
                    to 2^15
                    ;no significance lost
```

Before	After
(R1) = 000400	(R1) = 004000

Divide 071RSS



Operation: R, Rv1 ←R , Rv1 /(src)

Condition Codes: N: set if quotient <0; cleared otherwise, unless V is set, then undefined.
Z: set if quotient =0; cleared otherwise, unless V is set, then undefined.
V: set if source =0; set if the absolute value of the register is larger than the absolute value of the source. (In this case the instruction is aborted because the quotient would exceed 15 bits); or set if the quotient results in the most negative number (100000) and the XOR of the signs of the operands =0.
C: set if divide 0 attempted; cleared otherwise

Description: The 32-bit two's complement integer in R and Rv1 is divided by the source operand. The quotient is left in R; the remainder in Rv1. Division will be performed so that the remainder is of the same sign as the dividend. R must be even.

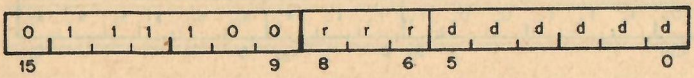
Example:

```
CLR R0
MOV #20001,R1
DIV #2,R0
```

Before	After	
(R0) = 000000	(R0) = 010000	Quotient
(R1) = 020001	(R1) = 000001	Remainder

XOR

Exclusive Or 074RDD



Operation: (dst)←Rv(dst)

Condition Codes:
N: set if the result <0; cleared otherwise
Z: set if result =0; cleared otherwise
V: cleared
C: unaffected

Description: The exclusive OR of the register and destination operand is stored in the destination address. Contents of register are unaffected. Assembler format is: XOR R,D

Example:

XOR R0,R2

Before	After
(R0) = 001234	(R0) = 001234
(R2) = 001111	(R2) = 000325

4.5.2 Logical Instructions

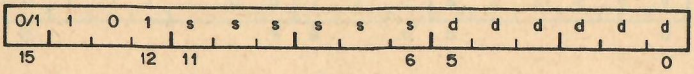
These instructions have the same format as the double operand arithmetic group. They permit operations on data at the bit level.

BIS
BIT
BIC

BIS
BISB

Bit Set

n5SSDD



Operation: (dst)←(src) v (dst)

Condition Codes:
N: set if high-order bit of result set, cleared otherwise
Z: set if result = zero; cleared otherwise
V: cleared
C: not affected

Description: Performs "Inclusive OR" operation between the source and destination operands and leaves the result at the destination address; that is, corresponding bits set in the source are set in the destination. The content of the destination are lost.

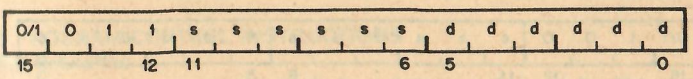
Example:

BIS R0,R1

Before	After
(R0) = 001234	(R0) = 001234
(R1) = 001111	(R1) = 001335
N Z V C	N Z V C
0 0 0 0	0 0 0 0

BIT
BITB

Bit Test n3SSDD



Operation: (dst)∧(src)

Condition Codes:
N: set if high-order bit of result set; cleared otherwise
Z: set if result =0; cleared otherwise
V: cleared
C: not affected

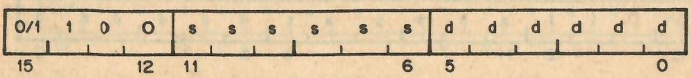
Description: Performs logical "and" comparison of the source and destination operands and modifies condition codes accordingly. Neither the source nor destination operands are affected. The BIT instruction may be used to test whether any of the corresponding bits that are set in the destination are also set in the source or whether all corresponding bits set in the destination are clear in the source.

Example:

```
BIT   #30,R3      ; test bits 3 and 4 of R3
                  ; to see if both are off
```

BIC
BICB

Bit Clear n4SSDD



Operation: (dst)←~(src)∧(dst)

Condition Codes:
N: set if high order bit of result set; cleared otherwise
Z: set if result =0; cleared otherwise
V: cleared
C: not affected

Description: Clears each bit in the destination that corresponds to a set bit in the source. The original contents of the destination are lost. The contents of the source are unaffected.

Example:

BIC R3,R4

Before	After
(R3) = 001234	(R3) = 001234
(R4) = 001111	(R4) = 000101
N Z V C	N Z V C
1 1 1 1	0 0 0 1

4.6 PROGRAM CONTROL INSTRUCTIONS

4.6.1 Branches

The instruction causes a branch to a location defined by the sum of the offset (multiplied by 2) and the current contents of the Program Counter if:

a) the branch instruction is unconditional

b) it is conditional and the conditions are met after testing the condition codes (status word).

The offset is the number of words from the current contents of the PC. Note that the current contents of the PC point to the word following the branch instruction.

Although the PC expresses a byte address, the offset is expressed in words. The offset is automatically multiplied by two to express bytes before it is added to the PC. Bit 7 is the sign of the offset. If it is set, the offset is negative and the branch is done in the backward direction. Similarly if it is not set, the offset is positive and the branch is done in the forward direction.

The 8-bit offset allows branching in the backward direction by 200_8 words (400_8 bytes) from the current PC, and in the forward direction by 177_8 words (376_8 bytes) from the current PC.

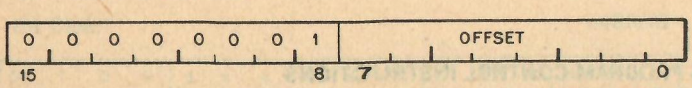
The PDP-11 assembler handles address arithmetic for the user and computes and assembles the proper offset field for branch instructions in the form:

Bxx loc

Where "Bxx" is the branch instruction and "loc" is the address to which the branch is to be made. The assembler gives an error indication in the instruction if the permissible branch range is exceeded. Branch instructions have no effect on condition codes.

BR

Branch (unconditional) 0004 loc



Operation: PC ← PC + (2 x offset)

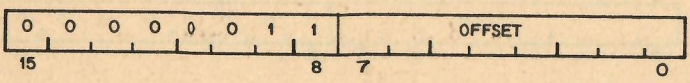
Description: Provides a way of transferring program control within a range of −128 to +127 words with a one word instruction.

Simple Conditional Branches

BEQ
BNE
BMI
BPL
BCS
BCC
BVS
BVC

BEQ

Branch on Equal (zero) 0014 offset



Operation: PC ← PC + (2 x offset) if Z = 1

Condition Codes: Unaffected

Description: Tests the state of the Z-bit and causes a branch if Z is set. As an example, it is used to test equality following a CMP operation, to test that no bits set in the destination were also set in the source following a BIT operation, and generally, to test that the result of the previous operation was zero.

Example:

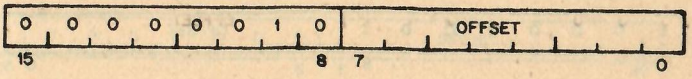
```
CMP   A,B          ; compare A and B
BEQ   C            ; branch if they are equal
```

will branch to C if A = B (A − B = 0)
and the sequence

```
ADD   A,B          ; add A to B
BEQ   C            ; branch if the result = 0
```

will breach to C if A + B = 0.

Branch Not Equal (Zero) 0010 offset



Operation: PC ← PC + (2 x offset) if Z = 0

Condition Codes: Unaffected

Description: Tests the state of the Z-bit and causes a branch if the Z-bit is clear. BNE is the complementary operation to BEQ. It is used to test inequality following a CMP, to test that some bits set in the destination were also in the source, following a BIT, and generally, to test that the result of the previous operation was not zero.

Example:

```
CMP   A,B        ; compare A and B
BNE   C          ; branch if they are not equal
```

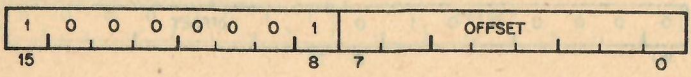
will branch to C if A ≠ B
and the sequence

```
ADD   A,B        ; add A to B
BNE   C          ; branch if the result is not equal
                   to 0
```

will branch to C if A + B ≠ 0

BMI

Branch on Minus 1004 offset



Operation: PC ← PC + (2 x offset) if N = 1

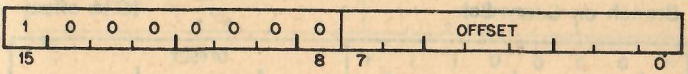
Condition Codes: Unaffected

Description: Tests the state of the N-bit and causes a branch if N is set. It is used to test the sign (most significant bit) of the result of the previous operation, branching if negative.

BPL

Branch on Plus

1000 offset

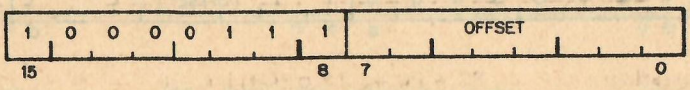


Operation: PC ← PC + (2 x offset) if N = 0

Description: Tests the state of the N-bit and causes a branch if N is clear. BPL is the complementary operation of BMI.

BCS

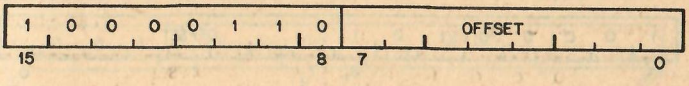
Branch on Carry Set 1034 offset



Operation: PC ← PC + (2 x offset) if C = 1

Description: Tests the state of the C-bit and causes a branch if C is set. It is used to test for a carry in the result of a previous operation.

Branch on Carry Clear 1030 offset

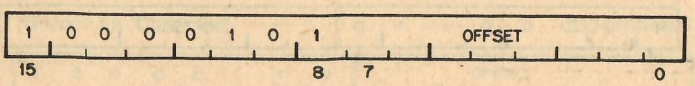


Operation: PC ← PC + (2 x offset) if C = 0

Description: Tests the state of the C-bit and causes a branch if C is clear. BCC is the complementary operation to BCS

BVS

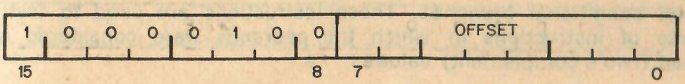
Branch on Overflow Set 1024 offset



Operation: PC ← PC + (2 x offset) if V = 1

Description: Tests the state of V bit (overflow) and causes a branch if the V bit is set. BVS is used to detect arithmetic overflow in the previous operation.

Branch on Overflow Clear 1020 offset



Operation: PC ← PC + (2 x offset) if V = 0

Description: Tests the state of the V bit and causes a branch if the V bit is clear. BVC is complementary operation to BVS.

Signed Conditional Branches

Particular combinations of the condition code bits are tested with the signed conditional branches. These instructions are used to test the results of instructions in which the operands were considered as a signed (two's complement) values.

Note that the sense of signed comparisons differs from that of unsigned comparisons in that in signed 16-bit, two's complement arithmetic the sequence of values is as follows:

largest	077777
	077776
positive	.
	.
	.
	000001
	000000
	177777
	177776
	.
negative	.
	.
	100001
smallest	100000

whereas in unsigned 16-bit arithmetic the sequence is considered to be

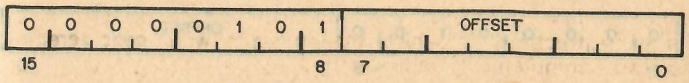
highest	177777
	.
	.
	.
	.
	.
	.
	000002
	000001
lowest	000000

The signed conditional branch instructions are:

BLT	BGE
BLE	BGT

BLT

Branch on Less Than (Zero) 0024 offset



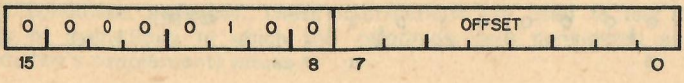
Operation: PC ← PC + (2 x offset) if N v V = 1

Description: Causes a branch if the "Exclusive Or" of the N and V bits are 1. Thus BLT will always branch following an operation that added two negative numbers, even if overflow occurred.

In particular, BLT will always cause a branch if it follows a CMP instruction operating on a negative source and a positive destination (even if overflow occurred). Further, BLT will never cause a branch when it follows a CMP instruction operating on a positive source and negative destination. BLT will not cause a branch if the result of the previous operation was zero (without overflow).

BGE

Branch on Greater than or Equal (zero) 0020 offset

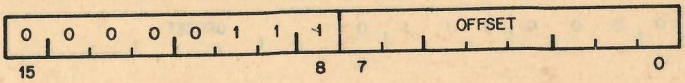


Operation: PC ← PC + (2 x offset) if N v V = 0

Description: Causes a branch if N and V are either both clear or both set. BGE is the complementary operation to BLT. Thus BGE will always cause a branch when it follows an operation that caused addition of two positive numbers. BGE will also cause a branch on a zero result.

BLE

Branch on Less than or Equal (zero) 0034 offset

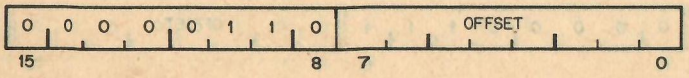


Operation: PC ← PC + (2 x offset) if Z v(N⊻V) = 1

Description: Operation is similar to BLT but in addition will cause a branch if the result of the previous operation was zero.

BGT

Branch on Greater Than (zero) 0030 offset



Operation PC ← PC + (2 x offset) if Z v(N ⊻ V) = 0

Description: Operation of BGT is similar to BGE, except BGT will not cause a branch on a zero result.

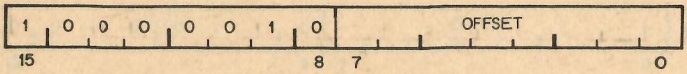
Unsigned Conditional Branches

The Unsigned Conditional Branches provide a means for testing the result of comparison operations in which the operands are considered as unsigned values.

BHI
BLOS
BHIS
BLO

BHI

Branch on Higher 1010 offset

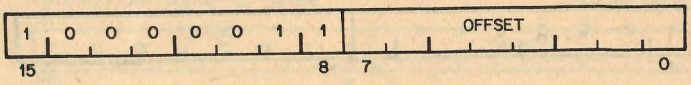


Operation: PC ← PC + (2 x offset) if C = 0 and Z = 0

Description: Causes a branch if the previous operation caused neither a carry nor a zero result. This will happen in comparison (CMP) operations as long as the source has a higher unsigned value than the destination.

BLOS

Branch on Lower or Same 1014 offset

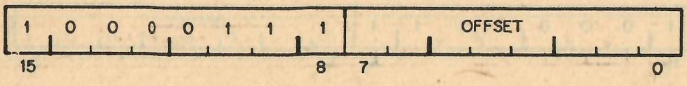


Operation: PC ← PC + (2 x offset) if C v Z = 1

Description: Causes a branch if the previous operation caused either a carry or a zero result. BLOS is the complementary operation to BHI. The branch will occur in comparison operations as long as the source is equal to, or has a lower unsigned value than the destination.

BLO

Branch on Lower 1034 offset

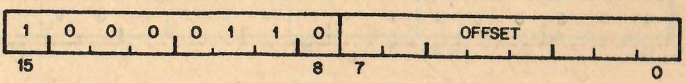


Operation: PC ← PC + (2 x offset) if C = 1

Description: BLO is same instruction as BCS. This mnemonic is included only for convenience.

BHIS

Branch on Higher or Same 1030 offset



Operation: PC ← PC + (2 x offset) if C = 0

Description: BHIS is the same instruction as BCC. This mnemonic is included only for convenience.

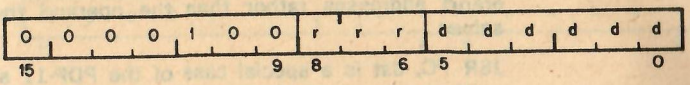
4.6.2 Subroutine Instructions

The subroutine call in the PDP-11 provides for automatic nesting of subroutines, reentrancy, and multiple entry points. Subroutines may call other subroutines (or indeed themselves) to any level of nesting without making special provision for storage or return addresses at each level of subroutine call. The subroutine calling mechanism does not modify any fixed location in memory, thus providing for reentrancy. This allows one copy of a subroutine to be shared among several interrupting processes. For more detailed description of subroutine programming see Chapter 7.

JSR
MARK
RTS

JSR

Jump to Sub Routine 004 reg. dst



Operation: (tmp)←(dst) (tmp is an internal processor register)

↓(SP)←reg push reg contents onto processor stack)

reg←PC (PC holds location following JSR; this address now put in reg

PC←(tmp) (PC now points to subroutine address)

Description: In execution of the JSR, the old contents of the specified register (the "LINKAGE POINTER") are automatically pushed onto the processor stack and new linkage information placed in the register. Thus subroutines nested within subroutines to any depth may all be called with the same linkage register. There is no need either to plan the maximum depth at which any particular subroutine will be called or to include instructions in each routine to save and restore the linkage pointer. Further, since all linkages are saved in a reentrant manner on the processor stack, execution of a subroutine may be interrupted, the same subroutine reentered and executed by an interrupt service routine. Execution of the initial subroutine can then be resumed when other requests are satisfied. This process (called nesting) can proceed to any level.

In both JSR and JMP instructions the destination address is used to load the program counter, PC. Thus for example a JSR in destination mode 1 for general register R1 (where (R1) = 100), will access a subroutine at location 100. This is effectively one level less of deferral than operate instructions such as ADD.

In the PDP-11/60, a JSR mode 0 will result in an "illegal" instruction and a trap through the trap vector address 4.

A subroutine called with a JSR reg,dst instruction can access the arguments following the call with either autoincrement addressing, (reg) +, (if arguments are accessed sequentially) or by indexed

addressing, X(reg), (if accessed in random order). These addressing modes may also be deferred, @(reg) + and @X(reg) if the parameters are operand addresses rather than the operand themselves.

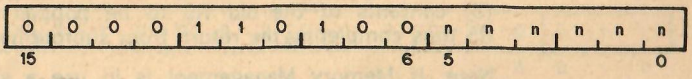
JSR PC, dst is a special case of the PDP-11 subroutine call suitable for subroutine calls that transmit parameters through the general registers. The SP and the PC are the only registers that may be modified by this call.

Another special case of the JSR instruction is JSR PC, @(SP) + which exchanges the top element of the processor stack and the contents of the program counter. Use of this instruction allows two routines to swap program control and resume operation when recalled where they left off. Such routines are called "co-routines."

Return from a subroutine is done by the RTS instruction. RTS reg loads the contents of reg into the PC and pops the top element of the processor stack into the specified register.

MARK

Mark 0064nn



Operation: SP←PC + 2xnn nn = number of parameters
PC←R5
R5←(SP)↑

Condition Codes: unaffected

Description: Used as part of the standard PDP-11 subroutine return convention. MARK facilitates the stack clean up procedures involved in subroutine exit. Assembler format is: MARK N

Example:

```
MOV   R5,—(SP)          ;place old R5 on stack
MOV   P1,—(SP)          ;place N parameters
MOV   P2,—(SP)          ;on the stack to be
                        ;used there by the
                        ;subroutine
MOV   PN,—(SP)
MOV   =MARKN,—(SP)      ;places the instruction
                        ;MARK N on the stack
MOV   SP ,R5            ;set up address at Mark N
                         instruction
JSR   PC,SUB            ;jump to subroutine
```

At this point the stack is as follows:

OLD R5
P1
PN
MARK N
OLD PC

And the program is at the address SUB which is the beginning of the subroutine.

```
SUB:                    ;execution of the subroutine it-
                         self
        RTS R5          ;the return begins: this causes
```

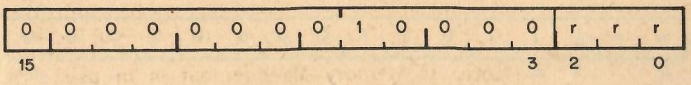
the contents of R5 to be placed in the PC which then results in the execution of the instruction MARK N. The contents of the old PC are placed in R5.

MARK N causes: (1) the stack pointer to be adjusted to point to the old R5 value; (2) the value now in R5 (the old PC) to be placed in the PC; and (3) contents of the old R5 to be popped into R5 thus completing the return from subroutine.

Note: If Memory Management is in use a stack must be in I and D spaces (Chapter 6) to execute the MARK instruction.

RTS

Return from Subroutine 00020 Reg



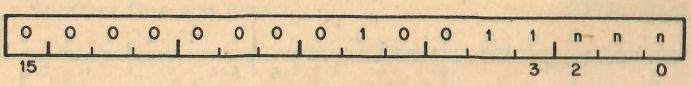
Operation: PC←reg
reg←(SP)↑

Description: Loads contents of reg into PC and pops the top element of the processor stack into the specified register.
Return from a non-reentrant subroutine is typically made through the same register that was used in its call. Thus, a subroutine called with a JSR PC, dst exits with a RTS PC and a subroutine called with a JSR R5, dst, may pick up parameters with addressing modes (R5)+, X(R5), or @X(R5) and finally exits, with an RTS R5.

4.6.3 Program Control Instructions

SPL
JMP
SOB

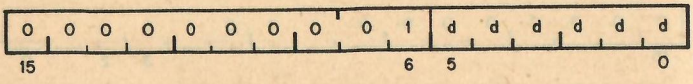
Set Priority Level 00023N



SPL is an illegal instruction in the PDP-11/60 and results in a processor trap through the trap vector address 10.

JMP

Jump 0001DD



Operation: PC←(dst)

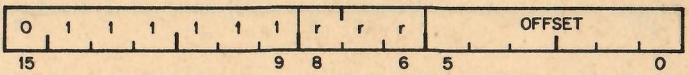
Condition Codes: not affected

Description: JMP provides more flexible program branching than provided with the branch instructions. Control may be transferred to any location in memory (no range limitation) and can be accomplished with the full flexibility of the addressing modes, with the exception of register mode 0. Execution of a jump with mode 0 will cause an "illegal" instruction condition and results in a processor trap through the trap vector address 4. (Program control cannot be transferred to a register.) Register deferred mode is legal and will cause program control to be transferred to the address held in the specified register. Note that instructions are word data and must therefore be fetched from an even-numbered address. A "boundary error" trap condition will result when the processor attempts to fetch an instruction from an odd address.

Deferred index mode JMP instructions permit transfer of control to the address contained in a selectable element of a table of dispatch vectors.

SOB

Subtract One and Branch 077R offset



Operation: R← R —1 if this result ≠ 0 then PC ← PC —(2x offset)

Condition Codes: unaffected

Description: The register is decremented. If it is not equal to 0, twice the offset is subtracted from the PC (now pointing to the following word). The offset is interpreted as a six bit positive number. This instruction provides a fast, efficient method of loop control. Assembler syntax is:

SOB R,A

Where A is the address to which transfer is to be made if the decremented R is not equal to 0. Note that the SOB instruction can not be used to transfer control in the forward direction.

NOTE

In the PDP-11/60, the SOB instruction may require more time than the sequence

DEC R
DNE.

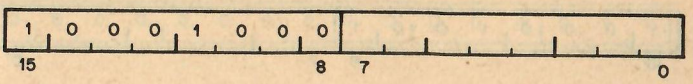
See Appendix B for timing information.

4.6.4 Traps

Trap Instructions—Provides for calls to emulators, I/O monitors, debugging packages, and user-defined interpreters. A trap is effectively an interrupt generated by software. When a trap occurs the contents of the current Program Counter (PC) and Program Status Word (PS) are pushed onto the processor stack and replaced by the contents of a two-word trap vector containing a new PC and new PS. The return sequence from a trap involves executing an RTI or RTT instruction which restores the old PC and old PS by popping them from the stack. Trap vectors are located permanently assigned fixed address.

TRAP
EMT
BPT
IOT
RTI
RTT

Emulator Traps 104000-104377



Operation:

↓(SP)←PS
↓(SP)←PC
PC←(30)
PS←(32)

Condition Codes:

N: loaded from trap vector
Z: loaded from trap vector
V: loaded from trap vector
C: loaded from trap vector

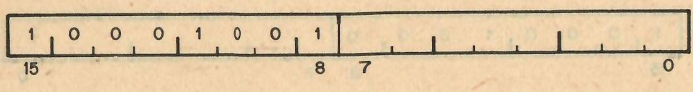
Description:

All operation codes from 104000 to 104377 are EMT instructions and may be used to transmit information to the emulating routine (e.g., function to be performed). The trap vector for EMT is at address 30. The new PC is taken from the word at address 30; the new central processor status (PS) is taken from the word at address 32.

Caution: EMT is used frequently by DIGITAL system software and is therefore not recommended for general use.

TRAP

Trap 104400 to 104777



Operation:

↓(SP)←PS
↓(SP)←PC
PC←(34)
PS←(36)

Condition Codes:

N: loaded from trap vector
Z: loaded from trap vector
V: loaded from trap vector
C: loaded from trap vector

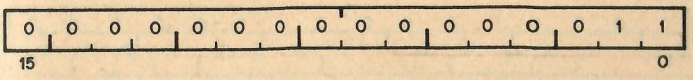
Description:

Operation codes from 104400 to 104777 are TRAP instructions. TRAPs and EMTs are identical in operation, except that the trap vector for TRAP is at address 34.

Note: Since DEC software makes frequent use of EMT, the TRAP instruction is recommended for general use.

BPT

Breakpoint Trap 000003



Operation:

↓(SP)←PS
↓(SP)←PC
PC←(14)
PC←(16)

Condition Codes:

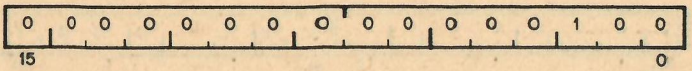
N: loaded from trap vector
Z: loaded from trap vector
V: loaded from trap vector
C: loaded from trap vector

Description:

Performs a trap sequence with a trap vector address of 14. Used to call debugging aids. The user is cautioned against employing code 000003 in programs run under these debugging aids.
(no information is transmitted in the low byte.)

IOT

I/O Trap 000004



Operation:

↓(SP)←PS
↓(SP)←PC
PC←(20)
PS←(22)

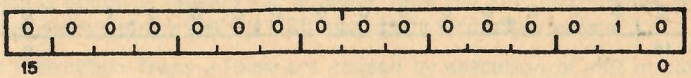
Condition Codes:

N: loaded from trap vector
Z: loaded from trap vector
V: loaded from trap vector
C: loaded from trap vector

Description:

Performs a trap sequence with a trap vector address of 20. Used to call the I/O Executive routine IOX in the paper tape software system, and for error reporting in the Disk Operating System.
(no information is transmitted in the low byte)

Return from Interrupt 000002



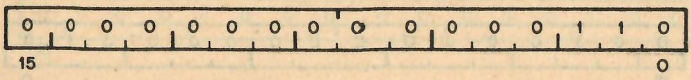
Operation: PC←(SP)↑
PS←(SP)↑

Condition Codes: N: loaded from processor stack
Z: loaded from processor stack
V: loaded from processor stack
C: loaded from processor stack

Description: Used to exit from an interrupt or TRAP service routine. The PC and PS are restored (popped) from the processor stack.

RTT

Return from Trap 000006



Operation: PC←(SP)↑
PS←(SP)↑

Condition Codes: N: loaded from processor stack
Z: loaded from processor stack
V: loaded from processor stack
C: loaded from processor stack

Description: This is the same as the RTI instruction except that it inhibits a trace trap, while RTI permits a trace trap. If a trace trap is pending, the first instruction after the RTT will be executed prior to the next "T" trap. In the case of the RTI instruction the "T" trap will occur immediately after the RTI.

Reserved Instruction Traps—These are caused by attempts to execute instruction codes reserved for future processor expansion (reserved instructions). Order codes not corresponding to any of the instructions described are considered to be reserved instructions.

XFC instructions are included if the UCS (User Control Store) option is not present and enabled. In these cases, the reserved instructions result in a macrotrap through vector address 10.

If the UCS option is present and enabled, both the XFC instructions and reserved instructions enter the UCS microcode for further action.

Illegal Instruction Traps—These are caused by execution of JSR or JMP instructions with destination mode 0 and result in a macrotrap through vector address 4.

Please note that some PDP-11 machines have placed this trap at vector address 10.

Overflow (Stack Limit) Traps—When instructions cause a stack address to exceed (go lower than) a limit set by the programmable Stack Limit Register (See Chapter 8), a Stack Violation can occur. If the instruction is JSR, TRAP, or has a destination address mode 1, 2, 4, or 6, a macro trap will occur. Non-altering memory references will not result in a trap, however.

There is a Yellow Zone (grace area) of 16 words below the Stack Limit which provides a warning to the program so that corrective steps can be taken. Operations that cause a Yellow Zone Violation are completed, then a trap through vector address 4 is effected. The error trap, which itself uses the stack, executes without causing an additional violation, unless the stack has entered the Red Zone.

A Red Zone Violation is a Fatal Stack Error. (Odd stack or non-existent stack are the other Fatal Stack Errors). When detected, the operation causing the error is aborted, the stack is repositioned to address 4, and a trap occurs. The old PC and PS are pushed into locations 0 and 2, and the new PC and PS are taken from locations 4 and 6.

Bus Error Traps—Bus Error Traps are:

1. Boundary Errors—attempts to reference instructions or word operands at odd addresses.
2. Time-Out Errors—attempts to reference addresses on the bus that made no response within 25 μsec. In general, these are caused by attempts to reference non-existent memory, and attempts to reference non-existent peripheral devices.

Bus error traps cause processor traps through the trap vector address 4.

Trace Trap—Trace Trap is enabled by bit 4 of the PS and causes processor traps at the end of instruction executions. The instruction that is executed after the RTT instruction that set the T-bit will proceed to completion and then cause a processor trap through the trap vector at address 14. Note that the trace is a system debugging aid and is transparent to the general programmer.

The following are special cases and are detailed in subsequent paragraphs.

1. The traced instruction cleared the T-bit.
2. The traced instruction set the T-bit.
3. The traced instruction caused an instruction trap.
4. The traced instruction caused a bus error trap.
5. The traced instruction caused a stack overflow trap.
6. The process was interrupted between the time the T-bit was set and the fetching of the instruction that was to be traced.
7. The traced instruction was a WAIT.
8. The traced instruction was a HALT.
9. The traced instruction was a Return from Trap.

Note: The traced instruction is the instruction after the one that sets the T-bit.

An instruction that cleared the T-bit—The trap will not occur at the end of execution of this instruction.

An instruction that set the T-bit—Since the T-bit was already set, setting it again has no effect. The trap will occur.

An instruction that caused an Instruction Trap—The instruction trap is sprung and the entire routine for the service trap is executed. If the service routine exits with an RTI, then a T-bit trap will occur at completion of the ATI. If the service routine exits with an RTT, then no T-bit trap will occur until the end of the next instruction. This assumes no other error traps occur.

An instruction that caused a Bus Error Trap—This is treated as an Instruction Trap. The only difference is that the error service is not as likely to exit with an RTI, so that the trace trap may not occur.

An instruction that caused a stack overflow—The instruction completes execution as usual—the Stack Overflow does not cause a trap. The Trace Trap Vector is loaded into the PC and PS, and the old PC and PS are pushed onto the stack. Stack Overflow occurs again, and this time the trap is made.

An interrupt between setting of the T-bit and fetch of the traced instruction—The entire interrupt service routine is executed and then the T-bit is set again by the exiting RTI and a T-bit trap occurs immediately.

Note that interrupts may be acknowledged immediately after the loading of the new PC and PS at the trap vector location. To lock out all interrupts, the PS at the trap vector should raise the processor priority to level 7.

A WAIT—The trap occurs immediately.

A HALT—The processor halts. When the continue key on the console is pressed, the instruction following the HALT is fetched and executed. Unless it is one of the exceptions noted above, the trap occurs immediately following execution.

A Return from Trap—The return from trap instruction either clears or sets the T-bit. It inhibits the trace trap. If the T-bit was set and RTT is the traced instruction the trap is delayed until completion of the next instruction.

Power Failure Trap —is a standard PDP-11 feature. Trap occurs whenever the AC power drops below 95 volts. Two milliseconds are then allowed for power down processing. Trap vector for power failure is at locations 24 and 26.

Other Traps are provided for specific functions associated with microbreak, floating point, memory management and memory parity errors. The functions and responses are discussed in separate chapters.

Trap priorities—in case multiple processor trap conditions occur simultaneously, the following order of priorities is observed (from high to low) in the PDP-11/60.

Traps are Serviced in the Following Order:

Abort Errors
- MICROBREAK (to UCS,Console, or macrotrap)
- UCS Parity Error
- Bus Error Traps (Odd Address error, Timeout)
- Stack Limit (Red Zone)
- Memory Management (KT) Abort
- Illegal Internal Address Reference
- Cache Parity Error (Abort Enabled)
- UNIBUS Timeout
- UNIBUS Memory Parity Error

- Illegal Instructions
- Reserved Instructions
- Trap Instructions
- Trace Trap
- Stack Limit (Yellow Zone)
- Cache Parity Error (Non-Abort)
- Power Fail
- Floating Point Exception
- Console Request
- UNIBUS Interrupt Request
- UCS Service Request
- WAIT Loop

Note that with simultaneous trap conditions, the macro code for the last condition serviced is the first to be executed.

Also, if more than one abort errors are asserted simultaneously, only the highest priority one will be serviced; the others will be lost.

The details on the trace trap process have been described in the trace trap operational description which includes cases in which an instruction being traced causes a bus error, instruction trap, or a stack overflow trap.

If a stack overflow is caused by the trap process in handling bus errors, instruction traps, or trace traps, the process is completed and then the stack overflow trap is sprung.

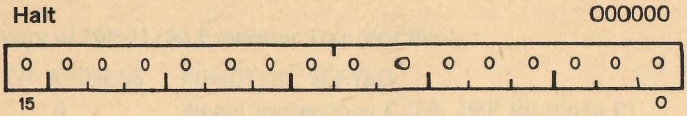
Summary of PDP-11/60 Processor Trap Vectors:

VECTOR ADDRESS	FUNCTION SERVED
4	Illegal Instructions (JSR, JMP for mode 0) Bus Errors (Odd Address Error, Timeout) Stack Limit (Red Zone, Yellow Zone) Illegal Internal Address Microbreak
10	Reserved Instruction XFC with UCS Disabled SPL, MTPS, MFPS, FADD, FSUB, FMUL, FDIV HALT in User Mode
14	Trace (T Bit)
20	IOT
24	Power Fail
30	EMT
34	TRAP
114	Cache Parity Error UNIBUS Memory Parity Error UCS Parity Error
244	Floating Point Exception
250	Memory Management (KT) Abort

4.7 MISCELLANEOUS

HALT
WAIT
RESET
MTPD
MTPI
MFPD
MFPI
MTPS
MFPS

HALT



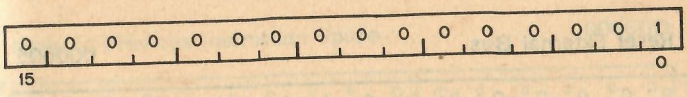
Condition Codes: not affected

Description: Causes the processor operation to cease. The console is given control of the bus. The console data lights display the contents of PC. The PC points to the next instruction to be executed. Pressing the continue key on the console causes processor operation to resume. No INIT signal is given.

Note: A halt issued in User Mode will generate a trap through vector adress 10.

Wait for Interrupt

000001



Condition Codes: not affected

Description: Provides a way for the processor to relinquish use of the bus while it waits for an interrupt. Having been given a WAIT command, the processor will not compete for bus use by fetching instructions or operands from memory. This permits higher transfer rates between a device and memory, since no processor-induced latencies will be encountered by bus requests from the device. In WAIT, as in all instructions, the PC points to the next instruction following the WAIT operation. Thus when the service routine executes an RTI instruction, at the end of the routine, the program will resume at the instruction following the WAIT. Note also that Floating Point, Power Fail, and Parity Traps will cause the processor to fall through the WAIT loop.

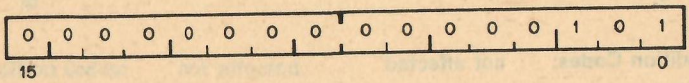
NOTE

Passive release of the bus will not cause the processor to fall through the WAIT loop.

A passive release occurs when a device requests the bus, but never transmits an Interrupt Vector when it receives a Bus Grant.

RESET

Reset External Bus 000005



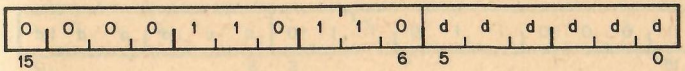
Condition Codes: not affected

Description: Sends INIT on the UNIBUS for 10 ms. All devices on the UNIBUS are reset to their state at power up.

In User Mode, the RESET instruction is treated as a no-op.

Within the PDP-11/60 processor, the Stack Limit and Memory Management Register, MMR0, are initialized.

Move to Previous Instruction Space 0066DD



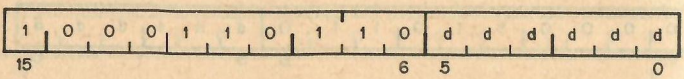
Operation: (temp)←(SP)↑
(dst)←(temp)

Condition Codes:
N: set if the source <0; otherwise cleared
Z: set if the source =0; otherwise cleared
V: cleared
C: unaffected

Description: The address of the destination operand is determined in the current address space. MTPI then pops a word off the current stack and stores that word in the destination address in the previous mode's I space (bits 13, 12 of PS).

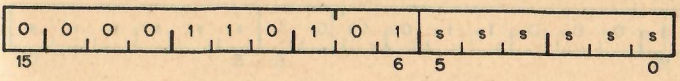
MTPD

Move to Previous Data Space 1066DD



MTPD is interpreted as MTPI in the PDP-11/60.

Move from Previous Instruction Space 0065SS



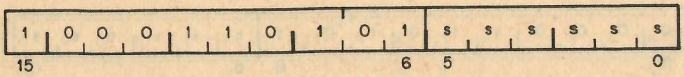
Operation: (temp) ← (src)
↓(SP) ←(temp)

Condition Codes: N: set if the source <0; otherwise cleared
Z: set if the source =0; otherwise cleared
V: cleared
C: unaffected

Description: This instruction is provided in order to allow inter-address space communication when the PDP11/45 is using the Memory Management unit. The address of the source operand is determined in the current address space. That is, the address is determined using the SP and memory pages determined by PS<15:14>. The address itself is then used in the previous I space (as determined by PS<13:12> to get the source operand. This operand is then pushed onto the current R6 stack.

MFPD

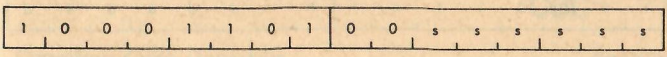
Move from Previous Data Space 1065SS



MFPD is interpreted as MFPI in the PDP-11/60.

MTPS

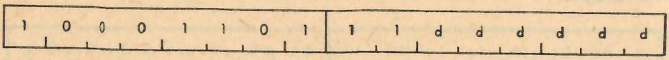
Move byte to Processor Status Word 1064SS



MTPS is not implemented in the PDP-11/60. A processor trap through vector address 10 occurs for reserved instruction.

MFPS

Move byte From Processor Status word 1067DD

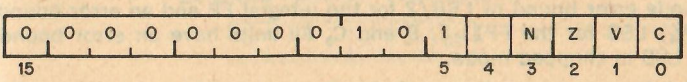


MFPS is not implemented in the PDP-11/60. A processor trap through vector address 10 occurs for reserved instruction.

4.8 CONDITION CODE OPERATORS

CLN SEN
CLZ SEZ
CLV SEV
CLC SEC
CCC SCC

Condition Code Operators 0002XX



Description: Set and clear condition code bits. Selectable combinations of these bits may be cleared or set together. Condition code bits corresponding to bits in the condition code operator (Bits 0-3) are modified according to the sense of bit 4, the set/clear bit of the operator. i.e. set the bit specified by bit 0, 1, 2 or 3, if bit 4 is a 1. Clear corresponding bits if bit 4 = 0.

Mnemonic	Operation	OP Code
CLC	Clear C	000241
CLV	Clear V	000242
CLZ	Clear Z	000244
CLN	Clear N	000250
SEC	Set C	000261
SEV	Set V	000262
SEZ	Set Z	000264
SEN	Set N	000270
SCC	Set all CC's	000277
CCC	Clear all CC's	000257
	Clear V and C	000243
	No Operation	000240

Combinations of the above set or clear operations may be ORed together to form combined instructions.

4.9 FLOATING POINT INSTRUCTIONS

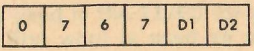
The instructions noted in Chapter 10 are performed in the basic PDP-11/60 processor. Because the basic processor uses a different data path and control mechanism than the optional FP11-E, the following differences exist.

1. On double precision ADD/SUB, the integral floating point will 2's complement before making an alignment shift of a negative operand; the FP11-E, B, and C all shift first. This difference results in a possible incompatibility in the LSB of the answer between the FP11-E, B and C (all of which are compatible) and the integral FP for both chopped and rounded modes. The difference also implies a rounded mode error bound of LSB/2 for the integral FP and an error bound of (¾) LSB for the FP11-E, B and C. All units have an error bound of 1 LSB in chopped mode.
2. Maintenance instructions noted in Paragraph 4.11 are for the optional FP11-E.

4.10 USER CONTROL STORE INSTRUCTIONS

The UCS (User Control Store) option for the PDP-11/60 utilizes the XFC (Extended Function Code) instruction. Details on use are contained in documentation associated with the UCS option.

Extended Function Code (USER)



This instruction provides dispatch information to the User Control Store or Extended Control Store. The D1 field is used for initial instruction group determination, with further instruction determination by D2 field or additional macro instruction words. If the option is not enabled, a trap through vector address 10 occurs.

4.11 MAINTENANCE INSTRUCTIONS

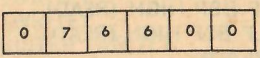
In the PDP-11/60, access to the large number of internal registers is provided by the MED (Maintenance, Exam and Deposit) instruction. Certain registers serve a dual purpose for an Error Log and these functions are noted in Chapter 11. Since this instruction provides access to processor control registers, extreme caution in use is necessary.

Maintenance instructions also exist within the Floating Point instructions for the optional FP11-E.

MED
LDUB
MNS
MPP
MAS

4.11.1 Base PDP-11/60 Maintenance Instruction

Maintenance, Exam and Dep



This instruction code is used in the 11/60 for a processor specific maintenance function. The first word is used as an escape with the CODE specifying the operation and address. The length of the instruction comes from a requirement that the console function be completely executed in one instruction time; otherwise normal console functions can interrupt. The use of a longer instruction also eases the need for dedicated internal registers. Service (Interrupt) requests are acknowledged at the end of this instruction but the internal state of the machine is not reset.

The instruction is executed only in Kernel Mode. Its main purpose is to allow error logging of internal registers and examination of internal registers for diagnostic purposes through the EXAM function. Instruction execution in User Mode will result in a trap to 10.

The instruction also allows through the DEP function an alteration of registers. For example, the cache memory might be "turned off" if sufficient errors occur.

The OPERATION CODE is specified and is register and operation dependent. The code directly benefits PDQ microcode and is shown in Appendix A.

R0, a general register, contains the information to be deposited or the results of an examination. The instruction is mainly for diagnostic purposes and failsafe features will not exist. The use of illegal OPERATION CODES will only be defined to the extent of completion of the instruction; No-op's will probably occur. Condition Codes are unaltered for this instruction.

The operation codes for the registers and function are noted below:

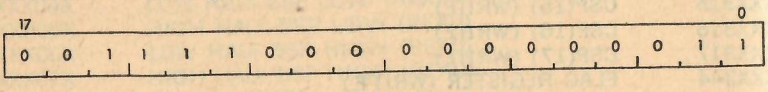
MED Operation Code and Function

MED CODE	REGISTER AND FUNCTION
XXX00X	LOW HALF ASP LOW (READ)
XXX01X	HIGH HALF ASP LOW (READ)
XXX02X	LOW HALF ASP HIGH (READ)
XXX03X	HIGH HALF ASP HIGH (READ)
XXX04X	LOW HALF BSP LOW (READ)
XXX05X	HIGH HALF BSP LOW (READ)
XXX06X	LOW HALF BSP HIGH (READ)
XXX07X	HIGH HALF BSP HIGH (READ)
XXX100	CSP(0) (READ)
XXX101	CSP(1) (READ)
XXX102	CSP(2) (READ)
XXX103	CSP(3) (READ)
XXX104	CSP(4) (READ)
XXX105	CSP(5) (READ)
XXX106	CSP(6) (READ)
XXX107	CSP(7) (READ)
XXX110	CSP(10) (READ)
XXX111	CSP(11) (READ)
XXX112	CSP(12) (READ)
XXX113	CSP(13) (READ)
XXX114	CSP(14) (READ)
XXX115	CSP(15) (READ)
XXX116	CSP(16) (READ)
XXX117	CSP(17) (READ)
XXX140	JAM (READ)
XXX141	SERVICE (READ)
XXX142	PHYSICAL BUS ADDRESS (READ)
XXX143	CUA (READ)
XXX144	FLAG REGISTER (READ)
XXX145	NOP
XXX146	REVISION REGISTER (READ)
XXX147	COUNT REGISTER (READ)
XXX152	DCS REGISTER #1 (READ)
XXX153	DCS REGISTER #2 (READ)
XXX154	CACHE INVALIDATE
XXX155	READ CACHE TAG
XXX20X	LOW HALF ASP LOW (WRITE)
XXX21X	HIGH HALF ASP LOW (WRITE)
XXX22X	LOW HALF ASP HIGH (WRITE)
XXX23X	HIGH HALF ASP HIGH (WRITE)
XXX24X	LOW HALF BSP LOW (WRITE)
XXX25X	HIGH HALF BSP LOW (WRITE)
XXX26X	LOW HALF BSP HIGH (WRITE)
XXX27X	HIGH HALF BSP HIGH (WRITE)
XXX300	CSP(0) (WRITE)
XXX301	CSP(1) (WRITE)
XXX302	CSP(2) (WRITE)
XXX303	CSP(3) (WRITE)
XXX304	CSP(4) (WRITE)

XXX305	CSP(5) (WRITE)
XXX306	CSP(6) (WRITE)
XXX307	CSP(7) (WRITE)
XXX310	CSP(10) (WRITE)
XXX311	CSP(11) (WRITE)
XXX312	CSP(12) (WRITE)
XXX313	CSP(13) (WRITE)
XXX314	CSP(14) (WRITE)
XXX315	CSP(15) (WRITE)
XXX316	CSP(16) (WRITE)
XXX317	CSP(17) (WRITE)
XXX344	FLAG REGISTER (WRITE)
XXX345	D REGISTER (WRITE)
XXX346	SHIFT REGISTER (WRITE)
XXX347	COUNTER(WRITE)
XXX350	NUA (WRITE)
XXX351	RES REGISTER (WRITE)
XXX352	INIT REGISTER (WRITE)
XXX353	NOP

4.11.2 Diagnostic Instructions

Load Microbreak Register 170003



This instruction causes the lower eight bits of general register 3 in the CPU to be loaded into the Microbreak register. LDUB can be used for the functions described in the following paragraphs, depending on the FMM bit (bit 04) in the program status word (FPS).

NOTE

The FMM bit in the status word is used to enable special maintenance logic. In order to set this bit, the CPU must be in Kernel mode.

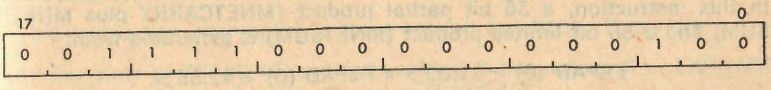
With the FMM bit set, the microprogram will be aborted through JAM μstate address 777 to the Ready state after the state specified by the address (next sequential μstate) in the Microbreak register is detected. If the interrupt enable bit (bit 14) of the floating-point processor status word is set, the CPU will trap to location 244. An exception code of 16 will be stored in the FEC (floating exception code) register. The contents of the FEC register can be transferred to the CPU by the STST (store status) instruction. A second function, available as a result of the LDUB instruction, is that the maintenance personnel can use the address match as a scope sync independent of the FMM bit. When the address matches the contents of the Microbreak register, the micro MATCH signal is present. This output is pin DC1 (slot 8 in the FNUA module) and is used as a scope sync to allow visual observation of events that occur during a particular μstate.

4.11.3 FP11-E Maintenance Instructions

The set of opcodes 170004 (MNS), 170005 (MPP), and 170007 (MAS) have been reserved as tools for debugging purposes.

This set together with the LDUB instruction (opcode 170003) should be used for diagnostic purposes only.

Maintenance Normalization Shift 170004



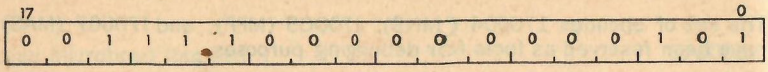
Instruction exercises the normalization (i.e., NORMK encoder, indirect control of shifter using NORMK, etc.), and rounding logic.

This Instruction Will:

1. Round the contents of FSPAD (0) in bit position 34 (02) for floating (double) precision number;
2. Left shift two places the results of the rounding operation. This action effectively drops the hidden bit;
3. Normalize the resulting number using the NORMK indirect control of the shifter. Result is left in FSPAD (1);
4. Adjust the exponent of AC0 (E(0)) to reflect normalization. Result is left in E(1).

MPP

Maintenance Partial Product 170005



The maintenance partial product (MPP) instruction is used for diagnostic purposes to test the multiplication network (MULNET).

In this instruction, a 36 bit partial product (MNETCARRY plus MNETSUM) and a 36 bit limited product (MNETSUM) is generated from:

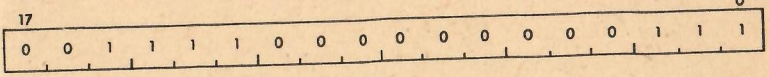
FSPAD (0) <31:03> • FSPAD (0) <42:35>.

The result is stored in FSPAD (1) <58:23> (MNETSUM), and FSPAD (2) <58:23> (MNETSUM plus MNETCARRY).

The exponents of FSPAD (1) and FSPAD (2) save the information needed to establish the contents of the most significant bit (AR<58>) of MNETSUM and MNETSUM plus MNETCARRY.

Maintenance Alignment Shift

170007



The maintenance alignment shift (MAS) instruction is used to test the alignment logic (i.e., quotient ROM, residue ROM, etc.), counter, and the alignment indirect control of the shifter.

This instruction accomplishes three related functions. These are:

1. Contents of FSPAD (0) are right shifted R bits, where R = EXP (AC0) MOD 11, and the result is stored in FSPAD (1).
2. NORMK logic encodes the number of leading zeros as a result of the shifting operation. This number (=4 LE. K .LE. 1) is then stored in E(1).
3. Counter is incremented P times, where P = INTG (EXP (AC0) / 11). Also, every time the counter is incremented the contents of E(2) are incremented by 1.

NOTE

Contents of AC1 and AC2 are modified by this instruction. However, the signs are left unchanged.

CHAPTER 5

MEMORY MANAGEMENT

5.1 GENERAL

5.1.1 Memory Management

This chapter describes the Memory Management unit of the PDP-11/60 Central Processor. The PDP-11/60 provides the hardware facilities necessary for complete memory management and protection. It is designed to be a memory management facility to allow memory size greater than 28K words and for multi-user, multi-programming systems where protection and relocation facilities are necessary.

5.1.2 Programming

The Memory Management hardware has been optimized towards a multi-programming environment and the processor can operate in two modes, Kernel and User. When in Kernel mode, the program has complete control and can execute all instructions. Monitors and supervisory programs would be executed in this mode.

When in User Mode, the program is prevented from executing certain instructions that could:

a) cause the modification of the Kernel program.
b) halt the computer.
c) use memory space assigned to the Kernel or other users.

In a multi-programming environment several user programs would be resident in memory at any given time. The task of the supervisory program would be: control the execution of the various user programs, manage the allocation of memory and peripheral device resources, and safeguard the integrity of the system as a whole by careful control of each user program.

In a multi-programming system, Memory Management provides the means for assigning pages (relocatable memory segments) to a user program and preventing that user from making any unauthorized access to those pages outside his assigned area. Thus, a user can effectively be prevented from accidental or willful destruction of any other user program or the system executive program.

Hardware implemented features enable the operating system to dynamically allocate memory upon demand while a program is being run. These features are particularly useful when running higher-level language programs, where, for example, arrays are constructed at execution time. No fixed space is reserved for them by the compiler. Lacking dynamic memory allocation capability, the program would have to calculate and allow sufficient memory space to accommodate the worst case. Memory Management eliminates this time-consuming and wasteful procedure.

5.1.3 Basic Addressing

The addresses generated by all PDP-11 Family Central Processors (CPs) are 18-bit direct byte addresses. Although the PDP-11 Family word length is 16 bits, the UNIBUS and CP addressing logic actually is 18 bits. Thus, while the PDP-11 word can only contain address references up to 32K words (64K bytes) the CP and UNIBUS can reference addresses up to 128K words (256K bytes). These extra two bits of addressing logic provide the basic framework for expanding memory references.

In addition to the word length constraint on basic memory addressing space, the uppermost 4K words of address space is always reserved for UNIBUS I/O device registers. In a PDP-11 memory configuration without Memory Management, all address references to the uppermost 4K words of 16-bit address space (160000-177777) are converted to full 18-bit references with bits 17 and 16 always set to 1. Thus, a 16-bit reference to the I/O device register at address 173224 is automatically internally converted to a full 18-bit reference to the register at address 773224.

5.1.4 Active Page Registers

The Memory Management Unit uses two sets of eight 32-bit Active Page Registers. An APR is actually a pair of 16-bit registers: a Page Address Register (PAR) and a Page Descriptor Register (PDR). These registers are always used as a pair and contain all the information needed to describe and relocate the currently active memory pages.

One set of APRs is used in Kernel mode, and the other in User mode. The choice of which set to be used is determined by the current CP mode contained in the Processor Status word.

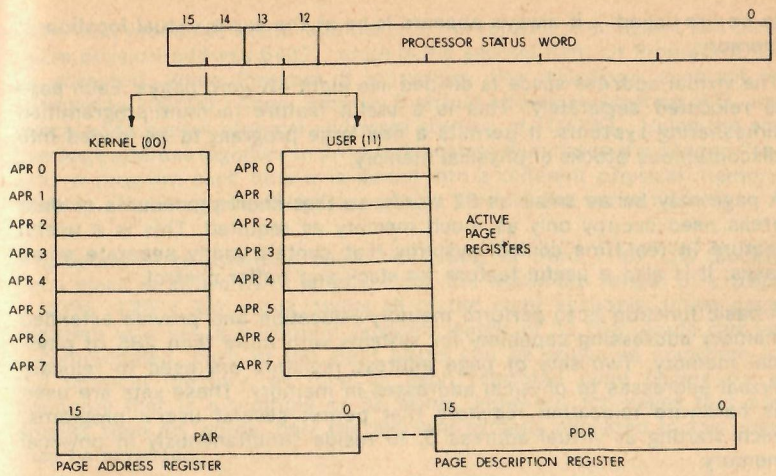


Figure 5-1 Active Page Registers

5.1.5 Capabilities Provided by Memory Management

Memory Size (words):	124K, max (plus 4K for I/O & registers)
Address Space:	Virtual (16 bits) Physical (18 bits)
Modes of Operation:	Kernel & User
Stack Pointers:	2 (one for each mode)
Memory Relocation:	
Number of Pages:	16 (8 for each mode)
Page Length:	32 to 4,096 words
Memory Protection:	no access read only read/write

5.2 RELOCATION

5.2.1 Virtual Addressing

When the Memory Management Unit is operating, the normal 16-bit direct byte address is no longer interpreted as a direct Physical Address (PA) but as a Virtual Address (VA) containing information to be used in constructing a new 18-bit physical address. The information contained in the Virtual Address (VA) is combined with relocation and description information contained in the Active Page Register (APR) to yield an 18-bit Physical Address (PA).

Because addresses are automatically relocated, the computer may be considered to be operating in virtual address space. This means that no matter where a program is loaded into physical memory, it will not have

to be "re-linked"; it always appears to be at the same virtual location in memory.

The virtual address space is divided into eight 4K-word pages. Each page is relocated separately. This is a useful feature in multi-programmed timesharing systems. It permits a new large program to be loaded into discontinuous blocks of physical memory.

A page may be as small as 32 words, so that short procedures or data areas need occupy only as much memory as required. This is a useful feature in real-time control systems that contain many separate small tasks. It is also a useful feature for stack and buffer control.

A basic function is to perform memory relocation and provide extended memory addressing capability for systems with more than 28K of physical memory. Two sets of page address registers are used to relocate virtual addresses to physical addresses in memory. These sets are used as hardware relocation registers that permit several user's programs, each starting at virtual address 0, to reside simultaneously in physical memory.

5.2.2 Program Relocation

The page address registers are used to determine the starting address of each relocated program in physical memory. Figure 5-2 shows a simplified example of the relocation concept.

Program A starting address 0 is relocated by a constant to provide physical address 6400_8.

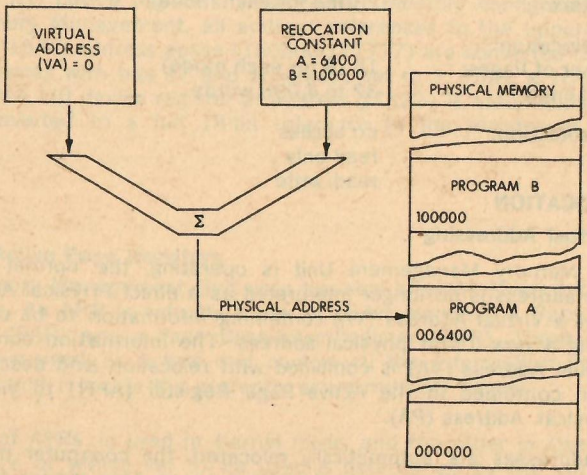


Figure 5-2 Simplified Memory Relocation Example

If the next processor virtual address is 2, the relocation constant will then cause physical address 6402_8, which is the second item of Program A, to be accessed. When Program B is running, the relocation constant is changed to 100000_8. Then, Program B virtual addresses starting at 0, are relocated to access physical addresses starting at 100000_8. Using the active page address registers to provide relocation eliminates the need to "relink" a program each time it is loaded into a different physical memory location. The program always appears to start at the same address.

A program is relocated in pages consisting of from 1 to 128 blocks. Each block is 32 words in length. Thus, the maximum length of a page is 4096 (128 x 32) words. Using all of the eight available active page registers in a set, a maximum program length of 32,768 words can be accommodated. Each of the eight pages can be relocated anywhere in the physical memory, as long as each relocated page begins on a boundary that is a multiple of 32 words. However, for pages that are smaller then 4K words, only the memory actually allocated to the page may be accessed.

The relocation example shown in Figure 5-3 illustrates several points about memory relocation.

a) Although the program appears to be in contiguous address space to the processor, the 32K-word physical address space is actually scattered through several separate areas of physical memory. As long as the total available physical memory space is adequate, a program can be loaded. The physical memory space need not be contiguous.

b) Pages may be relocated to higher or lower physical addresses, with respect to their virtual address ranges. In the example Figure 5-3, page 1 is relocated to a higher range of physical addresses, page 4 is relocated to a lower range, and page 3 is not relocated at all (even though its relocation constant is non-zero).

c) All of the pages shown in the example start on 32-word boundaries.

d) Each page is relocated independently. There is no reason why two or more pages could not be relocated to the same physical memory space. Using more than one page address register in the set to access the same space would be one way of providing different memory access rights to the same data, depending upon which part of a program was referencing that data.

Memory Units

Block:	32 words
Page:	1 to 128 blocks (32 to 4,096 words)
No. of pages:	8 per mode
Size of relocatable memory:	27,768 words, max (8 x 4,096)

a key feature in providing a fully protected environment for a time-shared multi-programming system.

A specific processor I/O address is assigned to each PAR and PDR of each set. Table 5-1 is a complete list of address assignment.

NOTE
UNIBUS devices cannot access PARs or PDRs

In a fully-protected multi-programming environment, the implication is that only a program operating in the Kernel mode would be allowed to write into the PAR and PDR locations for the purpose of mapping user's programs. However, there are no restraints imposed by the logic that will prevent User mode programs from writing into these registers. The option of implementing such a feature in the operating system, and thus explicitly protecting these locations from user's programs, is available to the system software designer.

Table 5-1 PAR/PDR Address Assignments

Kernel Active Page Registers			User Active Page Registers		
No.	PAR	PDR	No.	PAR	PDR
0	772340	772300	0	777640	777600
1	772342	772302	1	777642	777602
2	772344	772304	2	777644	777604
3	772346	772306	3	777646	777606
4	772350	772310	4	777650	777610
5	772352	772312	5	777652	777612
6	772354	772314	6	777654	777614
7	772356	772316	7	777656	777616

5.4.1 Page Address Registers (PAR)

The Page Address Register (PAR), shown in Figure 5-4, contains the 12-bit Page Address Field (PAF) that specifies the base address of the page.

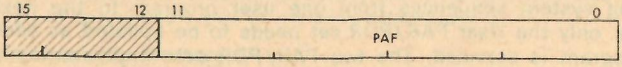


Figure 5-4 Page Address Register

Bits 15-12 are unused and reserved for possible future use.

The Page Address Register may be alternatively thought of as a relocation constant, or as a base register containing a base address. Either interpretation indicates the basic function of the Page Address Register (PAR) in the relocation scheme.

8.4.2 Page Descriptor Registers (PDR)

The Page Descriptor Register (PDR), shown in Figure 5-5, contains information relative to page expansion, page length, and access control.

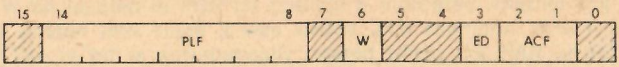


Figure 5-5 Page Descriptor Register

Access Control Field (ACF)
This 2-bit field, bits 2 and 1, of the PDR describes the access rights to this particular page. The access codes or "keys" specify the manner in which a page may be accessed and whether or not a given access should result in an abort of the current operation. A memory reference that causes an abort is not completed and is terminated immediately.

Aborts are caused by attempts to access non-resident pages, page length errors, or access violations, such as attempting to write into a read-only page. Traps are used as an aid in gathering memory management information.

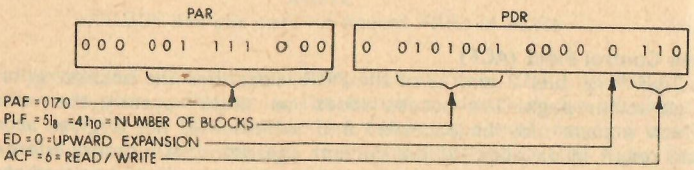
In the context of access control, the term "write" is used to indicate the action of any instruction which modifies the contents of any addressable word. A "write" is synonymous with what is usually called a "store" or "modify" in many computer systems. Table 5-2 lists the ACF keys and their functions. The ACF is written into the PDR under program control.

Table 5-2 Access Control Field Keys

AFC	Key	Description	Function
00	0	Non-resident	Abort any attempt to access this non-resident page
01	2	Resident read-only	Abort any attempt to write into this page.
10	4	(unused)	Abort all Accesses.
11	6	Resident read/ write	Read or Write allowed. No trap or abort occurs.

Expansion Direction (ED)
The ED bit located in PDR bit position 3 indicates the authorized direction in which the page can expand. A logic 0 in this bit (ED = 0) indicates the page can expand upward from relative zero. A logic 1 in this bit (ED = 1) indicates the page can expand downward toward relative zero. The ED bit is written into the PDR under program control. When the expansion direction is upward (ED = 0), the page length is increased by adding blocks with higher relative addresses. Upward expansion is usually specified for program or data pages to add more program or table space. An example of page expansion upward is shown in Figure 5-6.

When the expansion direction is downward (ED = 1), the page length is increased by adding blocks with lower relative addresses. Downward expansion is specified for stack pages so that more stack space can be added. An example of page expansion downward is shown in Figure 5-7.



NOTE:
To specify a block length of 42 for an upward expandable page, write highest authorized block no. directly into high byte of PDR. Bit 15 is not used because the highest allowable block number is 177_8.

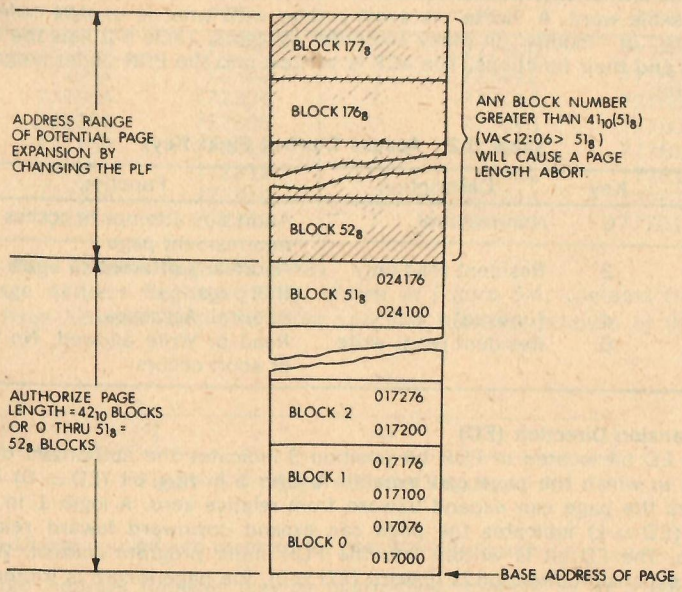


Figure 5-6 Example of an Upward Expandable Page

Written Into (W)
The W bit located in PDR bit position 6 indicates whether the page has been written into since it was loaded into memory. W = 1 is affirmative. The W bit is automatically cleared when the PAR or PDR of that page is written into. It can only be set by the control logic.

In disk swapping and memory overlay applications, the W bit (bit 6) can be used to determine which pages in memory have been modified by a user. Those that have been written into must be saved in their current form. Those that have not been written into (W = 0), need not be saved and can be overlayed with new pages, if necessary.

Page Length Field (PLF)
The 7-bit PLF located in PDR (bits 14-8) specifies the authorized length of the page, in 32-word blocks. The PLF holds block numbers from 0 to 177_8; thus allowing any page length from 1 to 128_{10} blocks. The PLF is written in the PDR under program control.

PLF for an Upward Expandable Page
When the page expands upward, the PLF must be set to one less than the intended number of blocks authorized for that page. For example, if 52_8 (42_{10}) blocks are authorized, the PLF is set to 51_8 (41_{10}) (Figure 5-6). The hardware compares the virtual address block number, VA (bits 12-6) with the PLF to determine if the virtual address is within the authorized page length.

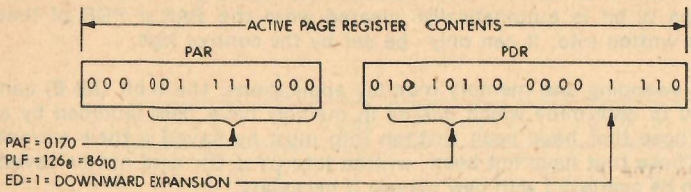
When the virtual address block number is less than or equal to the PLF, the virtual address is within the authorized page length. If the virtual address is greater than the PLF, a page length fault (address too high) is detected by the hardware and an abort occurs. In this case, the virtual address space legal to the program is non-contiguous because the three most significant bits of the virtual address are used to select the PAR/PDR set.

PLF for a Downward Expandable Page
The capability of providing downward expansion for a page is intended specifically for those pages that are to be used as stacks. In the PDP-11, a stack starts at the highest location reserved for it and expands downward toward the lowest address as items are added to the stack.

When the page is to be downward expandable, the PLF must be set to authorize a page length, in blocks, that starts at the highest address of the page. That is always Block 177_8. Refer to Figure 5-7, which shows an example of a downward expandable page. A page length of 42_{10} blocks is arbitrarily chosen so that the example can be compared with the upward expandable example shown in Figure 5-6.

NOTE

The same PAF is used in both examples. This is done to emphasize that the PAF, as the base address, always determines the lowest address of the page, whether it is upward or downward expandable.



To specify page length for a downward expandable page, write complement of blocks required into high byte of PDR.

In this example, a 42-block page is required.
PLF is derived as follows:

$42_{10} = 52_8$; two's complement = 126_8.

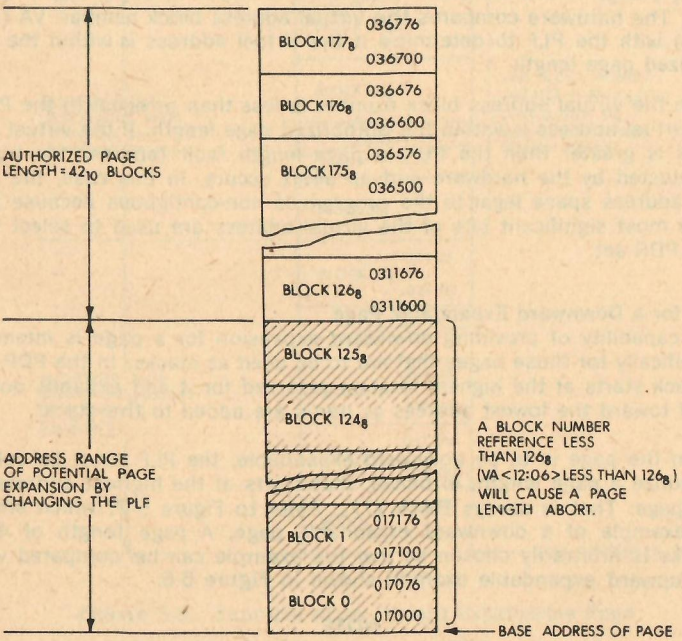


Figure 5-7 Example of a Downward Expandable Page

The calculations for complementing the number of blocks required to obtain the PLF is as follows:

MAXIMUM BLOCK NO.	MINUS	REQUIRED LENGTH	EQUALS	PLF
177_8	–	52_8	=	125_8
127_{10}	–	42_{10}	=	85_{10}

5.5 VIRTUAL & PHYSICAL ADDRESSES

The Memory Management Unit is located between the Central Processor Unit and the UNIBUS cache address lines. When Memory Management is enabled, the Processor ceases to supply address information to the UNIBUS/Cache. Instead, addresses are sent to the Memory Management Unit where they are relocated by various constants computed within the Memory Management Unit.

5.5.1 Construction of a Physical Address

The basic information needed for the construction of a Physical Address (PA) comes from the Virtual Address (VA), which is illustrated in Figure 5-8, and the appropriate APR set.

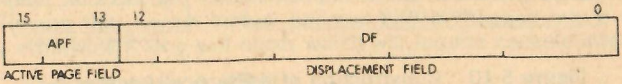


Figure 5-8 Interpretation of a Virtual Address

The Virtual Address (VA) consists of:

1. The Active Page Field (APF). This 3-bit field determines which of eight Active Page Registers (APR0-APR7) will be used to form the Physical Address (PA).

2. The Displacement Field (DF). This 13-bit field contains an address relative to the beginning of a page. This permits page lengths up to 4K words ($2^{13} = 8K$ bytes). The DF is further subdivided into two fields as shown in Figure 5-9.

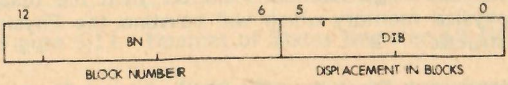


Figure 5-9 Displacement Field of Virtual Address

The Displacement Field (DF) consists of:

1. The Block Number (BN). This 7-bit field is interpreted as the block number within the current page.

2. The Displacement in Block (DIB). This 6-bit field contains the displacement within the block referred to by the Block Number.

The remainder of the information needed to construct the Physical Address comes from the 12-bit Page Address Field (PAF) (part of the Active Page Register) and specifies the starting address of the memory which that APR describes. The PAF is actually a block number in the physical memory, e.g. PAF = 3 indicates a starting address of 96, ($3 \times 32 = 96$) words in physical memory.

The formation of the Physical Address is illustrated in Figure 5-10.

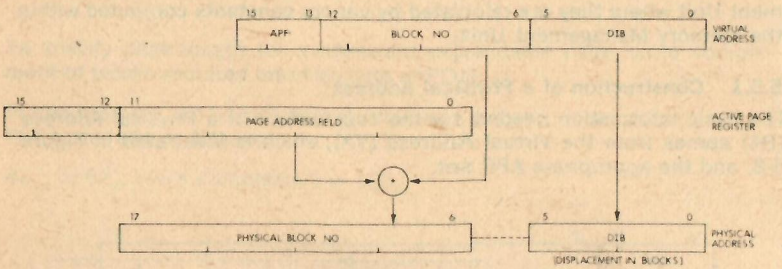


Figure 5-10 Construction of a Physical Address

The logical sequence involved in constructing a Physical Address is as follows:

1. Select a set of Active Page Registers depending on current mode.

2. The Active Page Field of the Virtual Address is used to select an Active Page Register (APR0-APR7).

3. The Page Address Field of the selected Active Page Register contains the starting address of the currently active page as a block number in physical memory.

4. The Block Number from the Virtual Address is added to the block number from the Page Address Field to yield the number of the block in physical memory which will contain the Physical Address being constructed.

5. The Displacement in Block from the Displacement Field of the Virtual Address is joined to the Physical Block Number to yield a true 18-bit Physical Address.

5.5.2 Determining the Program Physical Address

A 16-bit virtual address can specify up to 32K words, in the range from 0 to 177776_8 (word boundaries are even octal numbers). The three most significant virtual address bits designate the PAR/PDR set to be referenced during page address relocation. Table 8-3 lists the virtual address ranges that specify each of the PAR/PDR sets.

Table 5-3 Relating Virtual Address to PAR/PDR Set

Virtual Address Range	PAR/PDR Set
000000-17776	0
020000-37776	1
040000-57776	2
060000-77776	3
100000-117776	4
120000-137776	5
140000-157776	6
160000-177776	7

NOTE

Any use of page lengths less than 4K words causes holes to be left in the virtual address space.

5.6 STATUS REGISTERS

Aborts generated by the protection hardware are vectored through Kernel virtual location 250. Status Registers #0 and #2 are used to determine why the abort occurred. Note that an abort to a location which is itself an invalid address will cause another abort. Thus the Kernel program must insure that Kernel Virtual Address 250 is mapped into a valid address, otherwise a loop will occur which will require console intervention.

5.6.1 Status Register 0 (SR0)

SR0 contains abort error flags, memory management enable, plus other essential information required by an operating system to recover from an abort or service a memory management trap. The SR0 format is shown in Figure 5-11. Its address is 777 572.

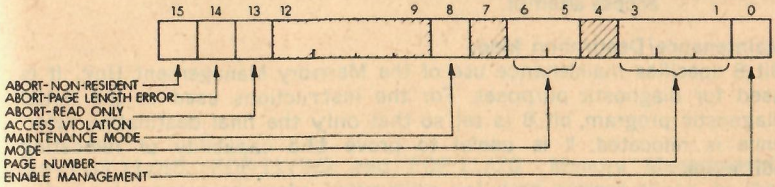


Figure 5-11 Format of Status Register #0 (SR0)

Bits 15-13 are the abort flags. They may be considered to be in a "priority queue" in that "flags to the right" are less significant and should be ignored. For example, a "non-resident" abort service routine would ignore page length and access control flags. A "page length" abort service routine would ignore an access control fault.

NOTE

Bit 15, 14, or 13, when set (abort conditions) cause the logic to freeze the contents of SR0 bits 1 to 6 and status register SR2. This is done to facilitate recovery from the abort.

Protection is enabled when an address is being relocated. This implies that either SR0, bit 0 is equal to 1 (Memory Management enabled) or that SR0, bit 8, is equal to 1 and the memory reference is the final one of a destination calculation (maintenance/destination mode).

Note that SR0 bits 0 and 8 can be set under program control to provide meaningful memory management control information. However, information written into all other bits is not meaningful. Only that information which is automatically written into these remaining bits as a result of hardware actions is useful as a monitor of the status of the memory management unit. Setting bits 15-13 under program control will not cause traps to occur. These bits, however, must be reset to 0 after an abort or trap has occurred in order to resume monitoring memory management. Bits 15-13 are not cleared by INIT.

Abort-Nonresident
Bit 15 is the "Abort-Nonresident" bit. It is set by attempting to access a page with an access control field (ACF) key equal to 0 or 4.

Abort—Page Length
Bit 14 is the "Abort-Page Length" bit. It is set by attempting to access a location in a page with a block number (virtual address bits 12-6) that is outside the area authorized by the Page Length Field (PFL) of the PDR for that page.

Abort-Read Only
Bit 13 is the "Abort-Read Only" bit. It is set by attempting to write in a "Read-Only" page having an access key of 2.

NOTE

There are no restrictions that any abort bits could not be set simultaneously by the same access attempt.

Maintenance/Destination Mode
Bit 8 specifies maintenance use of the Memory Management Unit. It is used for diagnostic purposes. For the instructions used in the initial diagnostic program, bit 8 is set so that only the final destination reference is relocated. It is useful to prove the capability of relocating addresses.

Mode of Operation
Bits 5 and 6 indicate the CPU mode (User or Kernel) associated with the page causing the abort. (Kernel = 00, User = 11).

Page Number
Bits 3-1 contain the page number of reference. Pages, like blocks, are numbered from 0 upwards. The page number bit is used by the error recovery routine to identify the page being accessed if an abort occurs.

Enable Relocation and Protection
Bit 0 is the "Enable" bit. When it is set to 1, all addresses are relocated

and protected by the memory management unit. When bit 0 is set to 0, the memory management unit is disabled and addresses are neither relocated nor protected.

5.6.2 Status Register 2 (SR2)

SR2 is loaded with the 16-bit Virtual Address (VA) at the beginning of each instruction fetch but is not updated if the instruction fetch fails. SR2 is read only; a write attempt will not modify its contents. SR2 is the Virtual Address Program Counter. Upon an abort, the result of SR0 bits 15, 14, or 13 being set, will freeze SR2 until the SR0 abort flags are cleared. The address of SR2 is 777 576.

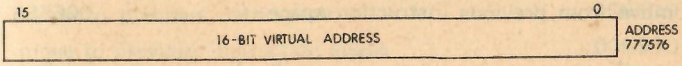


Figure 5-12 Format of Status Register 2 (SR2)

5.7 INSTRUCTIONS

Memory Management provides the ability to communicate between two spaces, as determined by the current and previous modes of the Processor Status word (PS).

Mnemonic	Instruction	Op Code
MFPI	move from previous instruction space	0065SS
MTPI	move to previous instruction space	0066DD
MFPD	move from previous data space	1065SS
MTPD	move to previous data space	1066DD

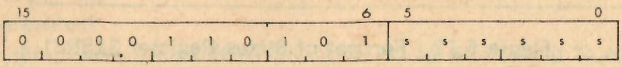
These instructions are directly compatible with the larger 11 computers.

The PDP-11/45, PDP-11/55 and PDP-11/70 Memory Management units implement a separate instruction and data address space. In the PDP-11/60, there is no differentiation between instruction or data space. The 2 instructions MFPD and MTPD (move to and from previous data space) execute identically to MFPI and MTPI.

MFPD
MFPI

move from previous data space 1065SS

move from previous instruction space 0065SS



Operation: (temp)←(src)
↓(SP)←(temp)

Condition Codes:
N: set if the source <0; otherwise cleared
Z: set if the source =0; otherwise cleared
V: cleared
C: unaffected

Description: This instruction pushes a word onto the current stack from an address in previous space, Processor Status (bits 13, 12). The source address is computed using the current registers and memory map.

Example:

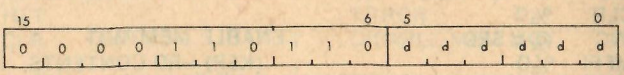
```
MFPI @ (R2)      R2 = 1000
                 1000 = 37526
```

The execution of this instruction causes the contents of (relative) 37526 of the previous address space to be pushed onto the current stack as determined by the PS (bits 15, 14).

MTPD
MTPI

move to previous data space 1066DD

move to previous instruction space 0066DD



Operation: (temp)←(SP)↑
(dst)←(temp)

Condition Codes:
N: set if the sourse <0; otherwise cleared
Z: set if the source =0; otherwise cleared
V: cleared
C: unaffected

Description: This instruction pops a word off the current stack determined by PS (bits 15, 14) and stores that word into an address in previous space PS (bits 13, 12). The destination address is computed using the current registers and memory map. An example is as follows:

Example:

MTPI @ (R2) R2 = 1000
1000 = 37526

The execution of this instruction causes the top word of the current stack to get stored into the (relative) 37526 of the previous address space.

MTPI AND MFPI, MODE 0, REGISTER 6 ARE UNIQUE IN THAT THESE INSTRUCTIONS ENABLE COMMUNICATIONS TO AND FROM THE PREVIOUS USER STACK.

```
; MFPI, MODE 0, NOT REGISTER 6

        MOV   #KM+PUM, PSW      ; KMODE, PREV USER
        MOV   #-1, -2(6)        ; MOVE -1 on kernel stack -2
        CLR   %0
        INC   @#SR0             ; ENABLE MEM MGT
        MFPI  %0                ; -(KSP)←R0 CONTENTS
```

The -1 in the kernel stack is now replaced by the contents of R0 which is 0.

```
; MFPI, MODE 0, REGISTER 6

        MOV   #UM+PUM, PSW
        CLR   %6                ; SET R16=0
        MOV   #KM+PUM, PSW      ; K MODE, PREV USER
        MOV   #-1, -2 (6)
        INC   @#SR0             ; ENABLE MEM MGT
        MFPI  %6                ; -(KSP)←R16 CONTENTS
```

The -1 in the kernel stack is now replaced by the contents of R16 (user stack pointer which is 0).

To obtain info from the user stack if the status is set to kernel mode, prev user, two steps are needed.

```
        MFPI  %6                ; get contents of R16=user pointer
        MFPI  @(6)+             ; get user pointer from kernel stack
                                ; use address obtained to get data
                                ; from user mode using the prev
                                ; mode
```

The desired data from the user stack is now in the kernel stack and has replaced the user stack address.

```
; MTPI, MODE 0      , NOT REGISTER 6

        MOV   #KM+PUM, PSW      ; KERNEL MODE, PREV USES
        MOV   #TAGX, (6)        ; PUT NEW PC ON STACK
        INC   @#SR0             ; ENABLE KT
        MTPI  %7                ; %7← (6)+
        HLT                     ; ERROR
TA6X:   CLR   @#SR0             ; DISABLE MEM MGT
```

The new PC is popped off the current stack and since this is mode 0 and not register 6 the destination is register 7.

```
; MTPI, MODE 0, REGISTER 6

        MOV   #UM+PUM, PSW      ; user mode, Prev User
        CLR   %6                ; set user SP=0 (R16)
        MOV   #KM+PUM, PSW      ; Kernel mode, prev user
        MOV   #−1, −(6)         ; MOVE −1 into K stack (R6)
        INC   @#SR0             ; Enable MEM MGT
        MTPI  %6                ; %16 ←(6)+
```

The 0 in R16 is now replaced with −1 from the contents of the kernel stack.

To place info on the user stack if the status is set to kernel mode, prev user mode, 3 separate steps are needed.

```
        MFPI  %6                ; Get content of R16=user pointer
        MOV   #DATA, −(6)       ; put data on current stack
        MTPI  @(6)+             ; @(6)+ [final address relocated]←
                                (R6)+
```

The data desired is obtained from the kernel stack then the destination address is obtained from the kernel stack and relocated through the previous mode.

CHAPTER 6

MEMORY

6.1 GENERAL

This chapter provides detailed information on the:

- Memory system
- Cache memory
- Memory error detection (parity)
- Memory registers (memory system error, cache control, hit/miss)

6.2 MEMORY SYSTEM

The memory system for the PDP-11/60 consists of a 2048 byte, high-speed, bipolar cache memory physically located within the processor, and up to 256K bytes of UNIBUS main memory, referred to as backing store (see figure 6-1).

The cache memory provides for rapid execution time of instructions, while the main memory provides bulk storage in a cost-effective ratio.

6.3 CACHE MEMORY

6.3.1 Introduction

A cache memory is a small, high-speed memory that maintains a copy of previously selected portions of main memory for faster re-access to instructions and data. A computer system, using a cache memory, appears the same as a conventional system with UNIBUS memory, except that the execution of programs is noticeably faster. The only difference is in system timing; there are no changes in programming. The operation is transparent to the user except for error support and cache control. The cache memory is physically located within the processor and is both a part of the processor and part of the backing store as shown in figure 6-2. The high-speed bipolar cache memory is synchronized with the processor and eliminates long bus transmission and access times associated with the backing store. Allocation mechanisms in the PDP-11/60 processor automatically and dynamically update the cache memory and extend the speed effect of cache across the entire backing store memory.

The statistics of program behavior make a cache system work. All of the data is stored in main memory; a copy of some of the data is stored in the cache. If, most of the time, the needed data is in the fast memory, the program will execute quickly, slowing down only when accesses must be made to main memory. Some semi-conductor/core systems achieve program execution speed by having the programmer control, in advance, which sections of the program should go in fast memory. The cache system achieves the same goal by automatically and dynamically loading cache memory in a way which gives a high probability that desired data will be in the fast memory.

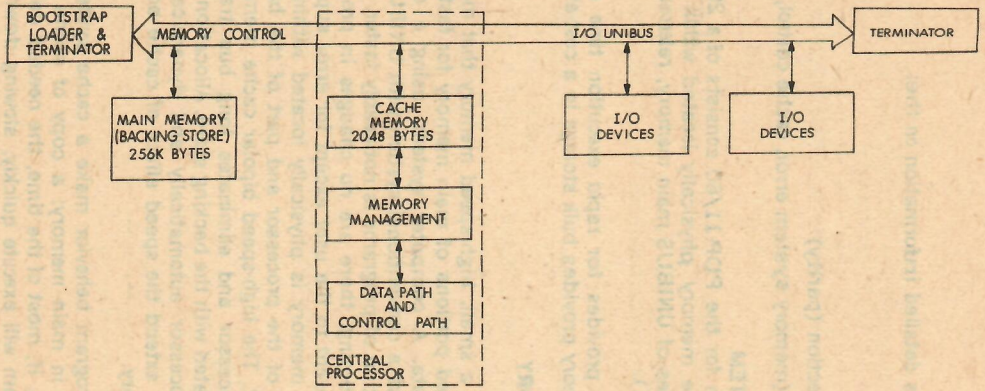


Figure 6-1 Simplified PDP-11/60 System

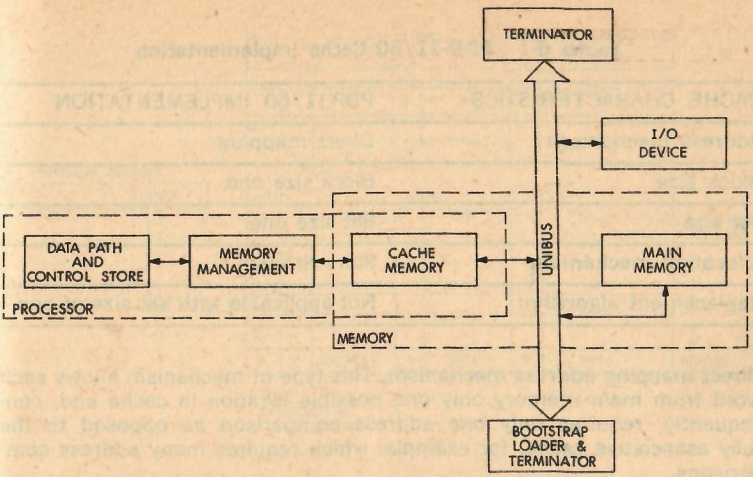


Figure 6-2 Cache Memory System Relationship

The principle of program locality states that programs have a tendency to make most accesses in the neighborhood of locations accessed in the recent past. Programs typically execute instructions in straight lines or small loops, with the next few accesses likely to be within a few words ahead of or behind the current location. Stacks grow and shrink from one end, with the next few accesses near the current top. Data elements are often scanned through sequentially. The cache makes use of this type of program behavior by keeping copies of recently used words.

From the standpoint of cost effectiveness, a cache system offers faster system speed for the cost of only a small quantity of fast memory plus associated logic, while main memory (backing store) can be economically implemented. An increase in system speed depends on the size and organization of cache, not on the type or speed of main memory. The user receives a very substantial speed improvement for a modest cost, and there are no programming changes. Although the exact speed improvement depends on the particular program, a judicious choice of architecture and algorithm will in general produce good results for all useful programs.

The fundamental concern is instruction execution speed. This is affected by the speeds of fast and slow memory and by the percentage of times memory references will find the data within the cache and, therefore, allow faster execution. When the needed data is found in the cache, a hit is said to occur. A miss occurs when the data is not in the cache. The cache system within the processor provides an additional advantage of lower UNIBUS utilization by the processor since read memory references that are hits do not access the UNIBUS. Consequently, the UNIBUS is more available for I/O device to memory transfers.

6.3.2 PDP-11/60 Cache Implementation

Different implementations of cache memory organization are possible. The specific implementation chosen for the PDP-11/60 is summarized in Table 6-1. The cache parameters listed in Table 6-1 are defined below.

Table 6-1 PDP-11/60 Cache Implementation

CACHE CHARACTERISTICS	PDP-11/60 IMPLEMENTATION
Address mechanism	Direct mapping
Block size	Block size one
Set size	Set size one
Allocation mechanism	Store through
Replacement algorithm	Not applicable with set size of one

Direct mapping address mechanism. This type of mechanism allows each word from main memory only one possible location in cache and, consequently, requires only one address comparison as opposed to the fully associative cache, for example, which requires many address comparisons.

Block size. The PDP-11/60 has a block size of one, which means that every time a fetch to the backing store occurs, only one word is fetched. One word is allocated to cache in the event of a miss.

Set size. The PDP-11/60 has a set size of one, which means that there is one unique location in cache for any given word from backing store. Consequently, if a miss occurs, only one cache location is available for the data to be written into.

Write-through. Write-through is the PDP-11/60 method of handling stale data in main memory. In the write-through method, the data stored in cache is immediately copied into main memory; main memory always has a valid copy of all data.

6.3.3 Cache Memory Data Format

Figure 6-3 shows the basic data format of the PDP-11/60 cache memory. The 2048 bytes of memory data are organized in 1024 words of 27 bits each. These 1024 words are index positions and are organized into a direct mapping cache. Bits 10 through 1 of the physical address access these index positions upon a memory reference. A complete address match requires a comparison of bits 17 through 11 of the physical address with the address information contained in the tag field of the index position. The tag field contains seven address bits, a valid bit, and a parity bit. The data field of the index position consists of two 8-bit bytes of data, each with byte parity.

6.3.4 Physical and Cache Address

Since the physical address space is 128K words, an address mapping technique is necessary to allow the 1K word cache to be mapped directly onto any one of the 128 blocks. The physical address is divided into a tag field, an index field, and a byte field as shown in figure 6-4.

The byte field selects the high or low byte. The index field determines which cache index position is used to store the copy of the data. This

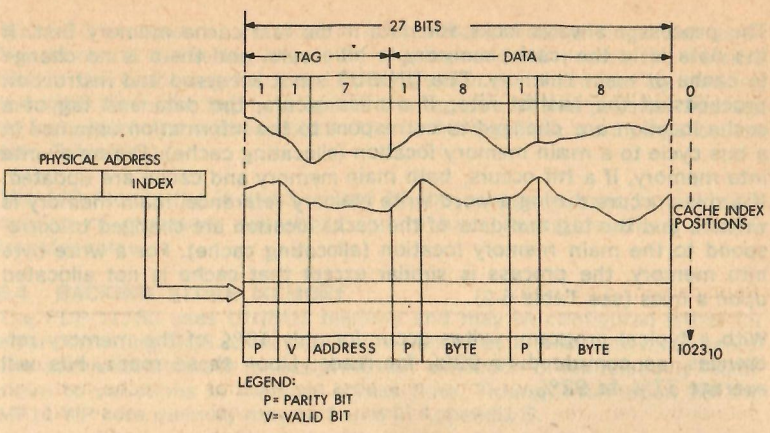


Figure 6-3 Cache Memory Data Format

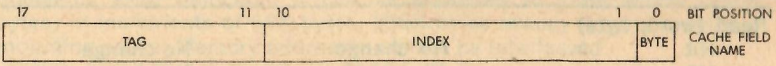


Figure 6-4 Physical Address Format

10-bit index field specifies one of 1024 index values and is the address of a 27-bit word in the cache (see figure 6-3).

For each of the index words, however, the remaining bits of the physical address can specify one of the 128 blocks. These bits comprise the tag field and are stored with the memory data in the cache index location. They prevent ambiguous determination of a specific physical address by uniquely specifying one of the 128 1K blocks.

Thus, addressing cache consists of applying the lower part of a physical address <10:1> against the 1K cache memory matrix and checking the higher order physical address <17:11> against the tag field of the index word obtained. If the tag field in the address matches the tag field stored with the data in the index word, the word obtained is the word specified by the physical address. This is designated a 'hit.' If the word is not the same (the fields do not match), it is designated as a 'miss.' On a processor write, a backing store reference occurs and the new data and tag portion of its physical address will be stored in the still accessed index position. This "allocation" keeps current data in the cache for processor use.

6.3.5 Processor Memory Reference

Cache memory within the PDP-11/60 (see figure 6-1) operates synchronously with processor memory references. Address information from the processor is translated to physical addresses by the Memory Management Unit (if enabled).

The processor always looks for data in the fast cache memory first. If the data is in the cache memory, a hit occurs, and there is no change to cache or main memory. The UNIBUS is not accessed and instruction proceeds at the fastest rate. If a miss occurs, the data and tag of a cache location are changed to correspond to the information obtained in a bus cycle to a main memory location (allocating cache). During a write into memory, if a hit occurs, both main memory and cache are updated. If a miss occurs during a word write memory reference, main memory is written, and the tag and data of the cache location are changed to correspond to the main memory location (allocating cache). For a write byte into memory, the process is similar except that cache is not allocated upon a miss (see Table 6-2).

With a typical program, writes occur for only 10% of the memory references, as compared to 90% for reads. Upon these reads, hits will average 77% to 92%.

Table 6-2: Hit or Miss Operations

Processor Operation	What Happens In Cache	What Happens In Main Memory
Read (word, byte)		
Hit	No change	No change
Miss	Allocated*	No change
Write (word)		
Hit	Updated**	Written Into
Miss	Allocated*	Written Into
Write (byte)		
Hit	Updated**	Written Into
Miss	No change	Written Into
NPR Operations	**What Happens In Cache**	**What Happens In Main Memory**
Read (word)		
Hit (not checked)	No change	No change
Miss (not checked)	No change	No change
Write (word or byte)		
Hit	Invalidated***	Written Into
Miss	No change	Written Into

* Allocated—The data and tag of the cache location is changed to correspond to the main memory location.

** Updated—The data in cache is changed to correspond to the data in main memory.

*** Invalidated—Valid bit in the cache word is cleared to show that the data is 'stale' and does not correspond to the data in main memory.

6.3.6 NPR Memory References

Exterior UNIBUS memory references (NPRs) that alter memory (write into memory) are monitored by the cache control logic. Physical address

bits 1-10 are used as an index to access the corresponding index position in cache. If the tag bits of the physical address match the address bits in the cache tag field, the index position is invalidated by clearing the valid bit in the tag field to 0. If the tag bits of the physical address do not match the address bits in the cache tag field, no change occurs (see Table 6-2).

The I/O monitoring is synchronized by the processor logic to maximize overlap of processor operations and to have negligible effect on I/O transfer rates.

6.4 BACKING STORE MEMORY

The PDP-11/60 uses UNIBUS memory and may be configured either for error-correcting MOS or for parity core memories in 64K byte increments. The cache architecture makes processor execution times relatively insensitive to variations in memory access time. Timing information for the MF11-WP core memory may be found in Appendix B.

6.4.1 MOS Memory

The PDP-11/60 can be supplied with the 32K-word (64K-byte) MF11S-KF memory system. This memory features single-bit error correction and double-bit error detection. It can be expanded to 128K words (256K bytes) in increments of 64K bytes. 256K bytes occupy 2 system units of mounting space. Memory addresses cannot be interleaved.

6.4.2 Core Memory

The PDP-11/60 can be supplied with the 32K-word (64K-byte) MF11-WP memory system. This parity memory can be expanded in 32K word (64K byte) increments to a full 256K byte backing store. Address interleave is standard for 128K- and 256K-byte sizes. 256K bytes occupy 4 system units of mounting space.

6.5 POWER FAILURE

When power is first applied, the Valid bit is cleared in all cache index values prior to any memory reference. First memory references are to the main memory. If power is lost, cache data will become invalid, but main memory with non-volatile core will have a correct copy of the data. If the machine contains MOS memory, with battery backup, a power fail will operate just as with core, provided the battery is functioning properly. If the battery is depleted, defective, or no battery backup is present, the machine will boot upon an automatic restart in Panel Lock mode. Otherwise, restart will be according to console switch setting.

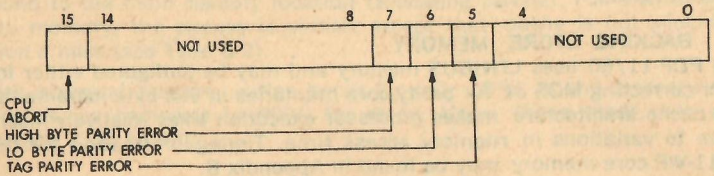
6.6 REGISTERS

The registers described in this section provide information about parity errors, memory status, and processor status. These hardware registers have program addresses in the top 4K words of physical address space (peripheral page).

Register	Address
Memory System Error	777744
Control	777746
Hit/Miss	777752

Some bit positions of the registers are not used (not implemented with hardware) and are indicated by cross-hatching in the illustrations. These bits are always read as zeros by the program. The Memory System Error Register is assembled from data within various error log registers and has certain restrictions. All of these registers are accessed by processor program execution or console action.

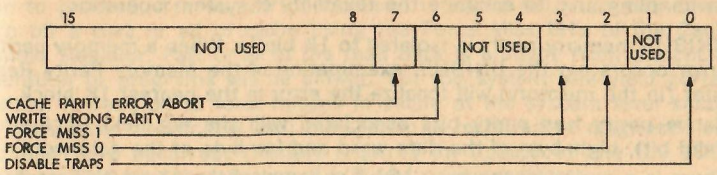
Memory System Error Register 777744



BIT	NAME	FUNCTION
15	CPU ABORT	Set if an error occurs that caused the processor to abort a cycle. The errors that cause this action are: UNIBUS Memory Parity Error; Cache Parity Error if the cache Parity Error Abort bit of the Cache Control Register is set; and User Control Store Parity Error.
14-8	Not Used	
7	HI BYTE	These bits are set for cache parity errors. The bits are set for parity errors in the high byte of data, the low byte of data, or the tag field (which includes the Valid bit), if the cycle is aborted. If the cycle is not aborted (cache parity error, abort bit of Cache Control Register is cleared and backing store references occur), all the bits (7, 6, 5) are set upon an error to aid compatibility with the PDP-11/70 system software. Then, if a cache parity error occurs, the Disable Traps bit of the Cache Control Register should be set to prevent the operating system from looping in the parity handler.
6	LO BYTE	
5	TAG PARITY	
4-0	Not Used	

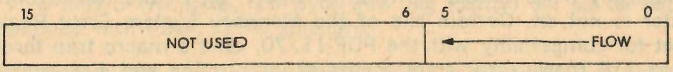
In the PDP-11/60, the Memory System Error Register is assembled from Error Log information (see Chapter 12) and is subject to the restrictions on the Error Log. The Error Log is stored, upon an error, in scratch pad registers used for floating point constants. If accurate error information is to be obtained, floating point instructions should not be executed between the parity error trap (location 114) and register access. The contents of this register are undefined if the last trap is not to location 114.

Cache Control Register 777746



BIT	NAME	FUNCTION
15-8	Not Used	
7	Cache Parity Error Abort	This bit is cleared on power up. It is set only during maintenance diagnostics and will cause an abort when a cache parity error occurs.
6	Write Wrong Parity	This bit is cleared on power up. It is used during maintenance diagnostics and, if set, will write wrong parity in the tag, high byte and low byte when cache is updated.
5-4	Not Used	
3-2	Force Miss	Setting these bits forces misses on reads to the cache and on attempts to invalidate the cache on NPR DATO references. Bit 3 forces misses on words 512 to 1023. Bit 2 forces misses on words 0 to 511. Setting both bits forces all cycles to main memory (degraded operation).
1	Not Used	
0	Disable Traps	Set by the cache parity error handler when it is desired to disable traps occurring as a result of non-fatal cache errors.

Hit/Miss Register 777752



This register indicates whether the six most recent references by the processor were hits or misses. A ONE (1) indicates a read hit; a ZERO (0) indicates a read miss or a write. The lower numbered bits are for the more recent cycles.

All the bits are read only. The bits are undetermined after a power up. They are not affected by a Console Start Clear.

6.7 PARITY

Parity is used extensively in the PDP-11/60 to insure the integrity of data handling and to enhance the reliability of system operation.

- UNIBUS memory parity is isolated to 1K blocks. When a memory parity error occurs on the UNIBUS, examination of the Memory Parity Register (in the memory) will localize the error to the nearest 1K block.
- Cache parity has parity bits associated with the tag field (including Valid bit), high byte of the data word and low byte of the data word.
- There is a parity bit for each 16-bit segment of the 48-bit WCS control word.

Software routines are used to log the occurrence of parity errors, to handle recovery from errors, and to provide information on system reliability and performance. Diagnostic software uses parity to isolate errors for rapid repair.

6.7.1 Basic Responses to Errors

The PDP-11/60 has two basic responses to parity errors:

1. The operation is aborted, an error log is generated concerning conditions at the point of error, and a macro trap is generated immediately; or
2. The operation continues, an error log cannot be generated, and a macro trap occurs at the end of the instruction. (The macro trap can be suppressed for cache errors if the DISABLE TRAPS bit of the Cache Control Register is set.)

The first response (abort) is necessary for UNIBUS memory parity errors, non-existent address time-out and User Control Store parity errors. In these instances, there is no way to continue or to reconstruct operation. For cache parity errors, the abort mode with its error log can be used for diagnostic purposes. This mode is enabled by setting the Cache Parity Error Abort bit of the Cache Control Register. Double errors within MOS memory will result in memory cycle abort with an immediate macro trap. The error correcting logic will correct the error and will set the single error bit in the MOS Memory Control and Status Registers. The register can be analyzed by system software to note degradation of memory operation. Continued operation depends upon the ability to obtain correct information. For cache parity errors, a reference to backing store can provide this information. This backing store reference occurs automatically if the Cache Parity Error Abort bit of the Cache Control Register is not set. Certain bits of the Memory System Error Register are set for compatibility with the PDP-11/70, and a macro trap through location 114 (parity error trap) occurs at instruction end. For error correcting codes in MOS memory, single errors are corrected in the memory to provide correct information.

6.7.2 System Responses to Errors

Extensive capabilities have been designed into the PDP-11/60 to allow recovery from cache parity errors, and to allow operation in a degraded mode if a section of the memory system is not operating properly. This type of operation is possible under program control by using the built-in control registers.

If data found in a location in cache does not have correct parity, a backing store reference can automatically occur to allow program execution to proceed. If a number of locations in cache fail, it is possible to turn off a part or all of cache using the Force Miss bits of the Cache Control Register. Part or all of the read data is brought from the cache; operation of programs will be slower, but the system will yield correct results. A decision to force misses in cache at the system level should be considered irrevocable until the system is restarted or diagnostic corrections have occurred. (Restart requires an update of the full 1024-word cache during the absence of I/O device intervention.)

If the macro trap after an automatic backing store reference takes too much system time, it can be suppressed by the Disable Traps bit of the Cache Control Register. This disable is also used in the service routine for the cache error to prevent endless traps.

If part of the main memory is not working, the Memory Management Unit can be used to map around the malfunctioning memory. Indication of main memory failure comes from the UNIBUS Memory Parity Error bit for single core memory failures and double MOS memory errors.

The error correcting logic will correct the error and will set the Single Error bit in the MOS Memory Control and Status Register. No direct macro program indication of an error is made. The Control and Status Register of the MOS memory does contain a Single Error bit that is set and remains set until cleared by program action. This register also contains a disable correction code bit to provide diagnostic determination of the exact error.

6.7.3 Cache Parity Error and Cache Control Register (CCR)

The system response to cache parity errors depends upon the state of the Cache Control Register bits CCR<07> (Cache Parity Error Abort) and CCR<00> (Disable Traps).

In most operations, CCR<07> and CCR<00> are zero. On a cache parity error, a trap will occur at the end of the current instruction. In this mode, where a cache parity error occurs, an internal control bit is set that will cause a trap through location 114, and a backing store reference occurs to obtain correct data. In the error handling routine, the CPU Abort bit (bit 15) in the Memory System Error Register, is examined. It will be zero, indicating that the instruction was not aborted. Bits 7, 6 and 5 (Hi Byte, Lo Byte and Tag Parity) will all be set for compatibility with PDP-11/70 software.

In certain situations (parity handler routine, for example), it is desirable to disable traps due to cache parity errors. This is done by making CCR<07> equal to zero and CCR<00> equal to one. In this mode, a cache parity error results in a backing store reference and no macro trap occurs.

If more detailed information about a cache parity error is required, as in a diagnostic, the current instruction is aborted. This mode occurs with CCR<07> set to one. When the error occurs, the memory reference cycle is aborted, an error log is constructed, and a macro trap through

location 114 occurs. The information in the error log includes exact parity error location to the address and byte level. When the Memory System Error Register is examined, it will be one, indicating that an instruction was aborted.

See Table 6-3 for a summary of operation.

Table 6-3 Actions Upon Cache Parity

Cache Control		Memory System Error		System Action
CCR<07>	CCR<00>	MSE<15>	MSE<07,06,05>	
0	0	0	All Set	Backing store references, trap through location 114 at instruction end.
0	1	0	All Set	Backing store references, no trap to location 114.
1	0	1	Set per Error	Abort current operation, construct error log, trap through location 114.
1	1	1	Set per Error	Abort current operation, construct error log, trap through location 114.

CHAPTER 7

PROGRAMMING TECHNIQUES

In order to produce programs which fully utilize the power and flexibility of the PDP-11, the reader should become familiar with the various programming techniques which are part of the basic design philosophy of the PDP-11. Although it is possible to program the PDP-11 along traditional lines such as "accumulator orientation" this approach does not fully exploit the architecture and instruction set of the PDP-11.

7.1 THE STACK

A "stack," as used on the PDP-11, is an area of memory set aside by the programmer for temporary storage or subroutine/interrupt service linkage. The instructions which facilitate "stack" handling are useful features not normally found in low-cost computers. They allow a program to dynamically establish, modify, or delete a stack and items on it. The stack uses the "last-in, first-out" concept; that is, various items may be added to a stack in sequential order and retrieved or deleted from the stack in reverse order. On the PDP-11, a stack starts at the highest location reserved for it and expands linearly downward to the lowest address as items are added to the stack.

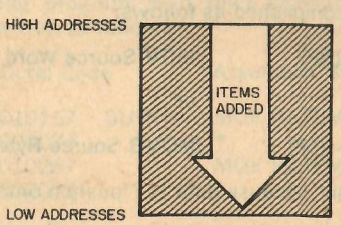


Figure 7-1 Stack Addresses

To keep track of the last item added to the stack (or "where we are" in the stack) a General Register always contains the memory address where the last item is stored in the stack. In the PDP-11 any register except Register 7 (the Program Counter-PC) may be used as a "stack pointer" under program control; however, instructions associated with subroutine linkage and interrupt service automatically use Register 6 (R6) as a hardware "Stack Pointer." For this reason R6 is frequently referred to as the system "SP".

Stacks in the PDP-11 may be maintained in either full word or byte units. This is true for a stack pointed to by any register except R6, which must be organized in full word units only.

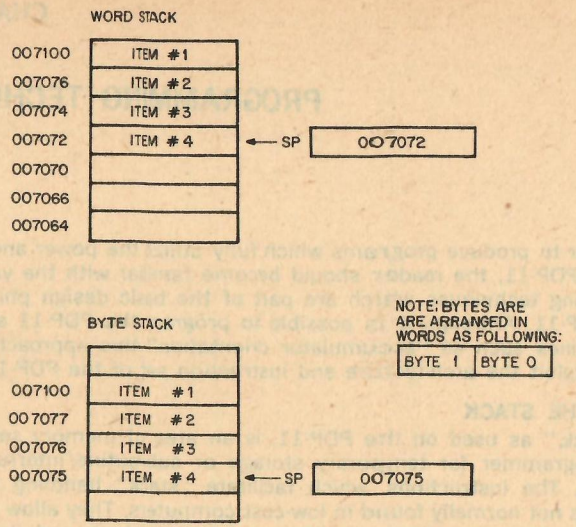


Figure 7-2 Word and Byte Stacks

Items are added to a stack using the autodecrement addressing mode with the appropriate pointer register. (See Chapter 3 for description of the autoincrement/decrement modes).

This operation is accomplished as follows:

```
MOV Source,—(SP)        ;MOV Source Word onto the stack

                        or

MOVB Source,—(SP)       ;MOVB Source Byte onto the stack
```

This is called a "push" because data is "pushed onto the stack."

To remove an item from a stack the autoincrement addressing mode with the appropriate SP is employed. This is accomplished in the following manner:

```
MOV(SP)+,Destination    ;MOV Destination Word off the stack

                        or

MOVB(SP)+,Destination   ;MOVB Destination Byte off the stack
```

Removing an item from a stack is called a "pop" for "popping from the stack." After an item has been "popped," its stack location is considered free and available for other use. The stack pointer points to the last-used location implying that the next (lower) location is free. Thus a stack may represent a pool of shareable temporary storage locations.

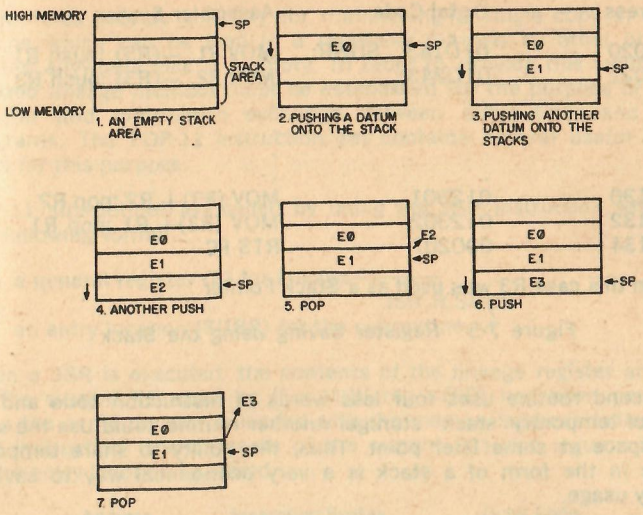


Figure 7-3 Illustration of Push and Pop Operations

As an example of stack usage consider this situation: a subroutine (SUBR) wants to use registers 1 and 2, but these registers must be returned to the calling program with their contents unchanged. The subroutine could be written as follows:

Address	Octal Code		Assembler Syntax
076322	010167	SUBR:	MOV R1,TEMP1 ; save R1
076324	000074		*
076326	010267		MOV R2,TEMP2 ;save R2
076330	000072		*
.	.		.
.	.		.
.	.		.
.	.		.
076410	016701		MOV TEMP1,R1 ;Restore R1
076412	000006		*
076414	016702		MOV TEMP2,R2 ; Restore R2
076416	000004		*
076420	000207		RTS PC
076422	000000		TEMP1: 0
076424	000000		TEMP2: 0

*Index Constants

Figure 7-4 Register Saving Without the Stack

OR: Using the Stack

```
Address     Octal Code      Assembler Syntax
010020      010143  SUBR:   MOV R1, —(R3) ;push R1
010022      010243          MOV R2, —(R3) ;push R2
.           .               .
.           .               .
.           .               .
.           .               .
010130      012301          MOV (R3)+,R2 ;pop R2
010132      012302          MOV (R3)+,R1 ;pop R1
010134      000207          RTS PC
```

Note: In this case R3 was used as a Stack Pointer

Figure 7-5 Register Saving using the Stack

The second routine uses four less words of instruction code and two words of temporary 'stack" storage. Another routine could use the same stack space at some later point. Thus, the ability to share temporary storage in the form of a stack is a very economical way to save on memory usage.

As a further example of stack usage, consider the task of managing an input buffer from a terminal. As characters come in, the terminal user may wish to delete characters from his line; this is accomplished very easily by maintaining a byte stack containing the input characters. Whenever a backspace is received a character is "popped" off the stack and eliminated from consideration. In this example, a programmer has the choice of "popping" characters to be eliminated by using either the MOVB (MOVE BYTE) or INC (INCREMENT) instructions.

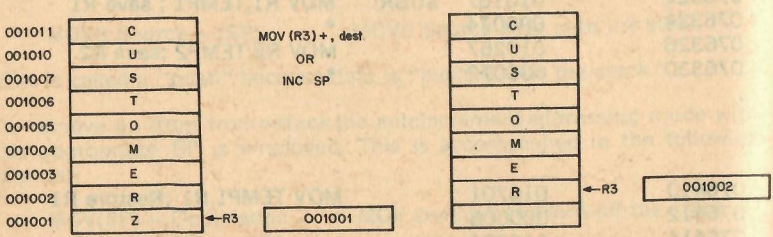


Figure 7-6 Byte Stack used as a Character Buffer

NOTE that in this case using the increment instruction (INC) is preferable to MOVB since it would accomplish the task of eliminating the unwanted character from the stack by readjusting the stack pointer without the need for a destination location. Also, the stack pointer (SP) used in this example cannot be the system stack pointer (R6) because R6 may only point to word (even) locations.

7.2 SUBROUTINE LINKAGE

7.2.1 Subroutine Calls

Subroutines provide a facility for maintaining a single copy of a given routine which can be used in a repetitive manner by other programs located anywhere else in memory. In order to provide this facility, generalized linkage methods must be established for the purpose of control transfer and information exchange between subroutines and calling programs. The PDP-11 instruction set contains several useful instructions for this purpose.

PDP-11 subroutines are called by using the JSR instruction which has the following format.

a general register (R) for linkage
an entry location (SUBR) for the subroutine

```
JSR R,SUBR
```

When a JSR is executed, the contents of the linkage register are saved on the system R6 stack as if a MOV reg,—(SP) had been performed. Then the same register is loaded with the memory address following the JSR instruction (the contents of the current PC) and a jump is made to the entry location specified.

Address	Assembler Syntax	Octal Code
001000	JSR R5, SUBR	004567
001002	index constant for SUBR	000060
001004		
001064	SUBR: MOV A,B	01nnmm

Figure 7-7 JSR using R5

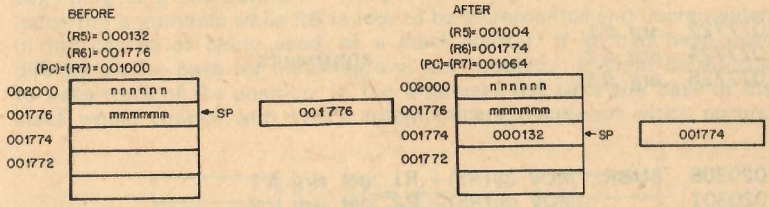


Figure 7-8 JSR

Note that the instruction JSR R6,SUBR is not normally considered to be a meaningful combination.

7.2.2 Argument Transmission

The memory location pointed to by the linkage register of the JSR instruction may contain arguments or addresses of arguments. These arguments may be accessed from the subroutine in several ways. Using Register 5 as the linkage register, the first argument could be obtained by using the addressing modes indicated by (R5),(R5)+,X(R5) for actual data, or @(R5)+, etc. for the address of data. If the autoincrement

mode is used, the linkage register is automatically updated to point to the next argument.

Figures 7-9 and 7-10 illustrate two possible methods of argument transmission.

```
Address Instructions and Data

010400              JSR R5, SUBR
010402              Index constant for SUBR      SUBROUTINE CALL
010404              arg #1                       ARGUMENTS
010406              arg #2
.                   .
.                   .
.                   .
.                   .
.                   .
020306   SUBR:      MOV (R5)+,R1    ;get arg #1
020310              MOV (R5)+,R2    ;get arg #2 Retrieve Arguments
                                    from SUB
```

Figure 7-9 Argument Transmission-Register Autoincrement Mode

```
Address   Instructions and Data

010400    JSR R5, SUBR
010402    Index constant for SUBR     SUBROUTINE CALL
010404    077722                      Address of arg #1
010406    077724                      Address of arg #2
010410    077726                      Address of arg #3
.         .                           .
.         .                           .
.         .                           .
077722    arg #1
077724    arg #2                      arguments
077726    arg #3
.         .                           .
.         .                           .
.         .                           .
020306    SUBR:  MOV @(R5)+,R1  ;get arg #1
020301           MOV @(R5)+,R2  ;get arg #2
```

Figure 7-10 Argument Transmission-Register Autoincrement Deferred Mode

Another method of transmitting arguments is to transmit only the address of the first item by placing this address in a general purpose register. It is not necessary to have the actual argument list in the same general area as the subroutine call. Thus a subroutine can be called to work on data located anywhere in memory. In fact, in many cases, the operations performed by the subroutine can be applied directly to the data located on or pointed to by a stack without the need to ever actually move this data into the subroutine area.

```
Calling Program: MOV     POINTER, .R1
                 JSR     PC,SUBR

SUBROUTINE       ADD     (R1) + ,(R1)    ;Add item #1 to item #2, place
                                         result in item #2, R1 points
                                         to item #2 now
                         etc.
                         or

                 ADD     (R1),2(R1)      ;Same effect as above except
                                         that R1 still points to item #1
                                         etc.
```

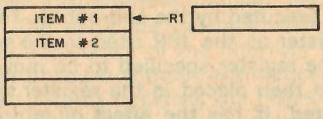


Figure 7-11 Transmitting Stacks as Arguments

Because the PDP-11 hardware already uses general purpose register R6 to point to a stack for saving and restoring PC and PS (processor status word) information, it is quite convenient to use this same stack to save and restore intermediate results and to transmit arguments to and from subroutines. Using R6 in this manner permits extreme flexibility in nesting subroutines and interrupt service routines.

Since arguments may be obtained from the stack by using some form of register indexed addressing, it is sometimes useful to save a temporary copy of R6 in some other register which has already been saved at the beginning of a subroutine. In the previous example R5 may be used to index the arguments while R6 is free to be incremented and decremented in the course of being used as a stack pointer. If R6 had been used directly as the base for indexing and not "copied," it might be difficult to keep track of the position in the argument list since the base of the stack would change with every autoincrement/decrement which occurs.

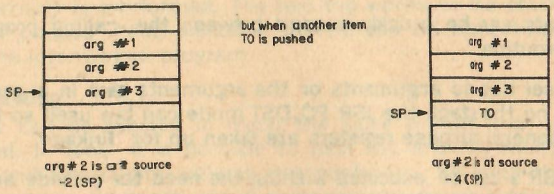


Figure 7-12 Shifting Indexed Base

However, if the contents of R6 (SP) are saved in R5 before any arguments are pushed onto the stack, the position relative to R5 would remain constant.

mode is used, the linkage register is automatically updated to point to the next argument.

Figures 7-9 and 7-10 illustrate two possible methods of argument transmission.

```
Address  Instructions and Data

010400           JSR R5, SUBR
010402           Index constant for SUBR      SUBROUTINE CALL
010404           arg #1                       ARGUMENTS
010406           arg #2
.                .
.                .
.                .
.                .
.                .
020306   SUBR:   MOV (R5)+,R1    ;get arg #1
020310           MOV (R5)+,R2    ;get arg #2 Retrieve Arguments
                                 from SUB
```

Figure 7-9 Argument Transmission-Register Autoincrement Mode

```
Address  Instructions and Data

010400   JSR R5, SUBR
010402   Index constant for SUBR     SUBROUTINE CALL
010404   077722                      Address of arg #1
010406   077724                      Address of arg #2
010410   077726                      Address of arg #3
.        .                           .
.        .                           .
.        .                           .
077722   arg #1
077724   arg #2                      arguments
077726   arg #3
.        .                           .
.        .                           .
.        .                           .
020306   SUBR:   MOV @(R5)+,R1  ;get arg #1
020301           MOV @(R5)+,R2  ;get arg #2
```

Figure 7-10 Argument Transmission-Register Autoincrement Deferred Mode

Another method of transmitting arguments is to transmit only the address of the first item by placing this address in a general purpose register. It is not necessary to have the actual argument list in the same general area as the subroutine call. Thus a subroutine can be called to work on data located anywhere in memory. In fact, in many cases, the operations performed by the subroutine can be applied directly to the data located on or pointed to by a stack without the need to ever actually move this data into the subroutine area.

```
Calling Program: MOV     POINTER, R1
                 JSR     PC,SUBR

SUBROUTINE       ADD     (R1)+,(R1)    ;Add item #1 to item #2, place
                                       result in item #2, R1 points
                                       to item #2 now
                         etc.
                         or

                 ADD     (R1),2(R1)    ;Same effect as above except
                                       that R1 still points to item #1
                                       etc.
```

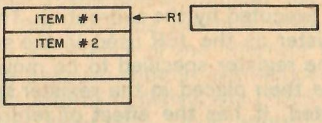


Figure 7-11 Transmitting Stacks as Arguments

Because the PDP-11 hardware already uses general purpose register R6 to point to a stack for saving and restoring PC and PS (processor status word) information, it is quite convenient to use this same stack to save and restore intermediate results and to transmit arguments to and from subroutines. Using R6 in this manner permits extreme flexibility in nesting subroutines and interrupt service routines.

Since arguments may be obtained from the stack by using some form of register indexed addressing, it is sometimes useful to save a temporary copy of R6 in some other register which has already been saved at the beginning of a subroutine. In the previous example R5 may be used to index the arguments while R6 is free to be incremented and decremented in the course of being used as a stack pointer. If R6 had been used directly as the base for indexing and not "copied," it might be difficult to keep track of the position in the argument list since the base of the stack would change with every autoincrement/decrement which occurs.

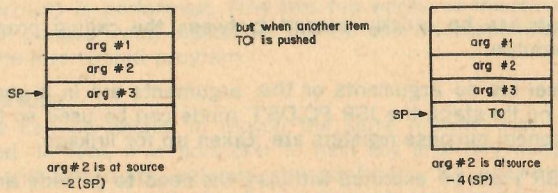


Figure 7-12 Shifting Indexed Base

However, if the contents of R6 (SP) are saved in R5 before any arguments are pushed onto the stack, the position relative to R5 would remain constant.

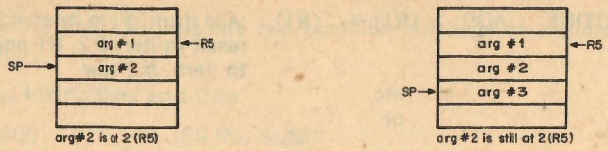


Figure 7-13 Constant Index Base Using "R6 Copy"

7.2.3 Subroutine Return

In order to provide for a return from a subroutine to the calling program an RTS instruction is executed by the subroutine. This instruction should specify the same register as the JSR used in the subroutine call. When executed, it causes the register specified to be moved to the PC and the top of the stack to be then placed in the register specified. Note that if an RTS PC is executed, it has the effect of returning to the address specified on the top of the stack.

Note that the JSR and the JMP instructions differ in that a linkage register is always used with a JSR; there is no linkage register with a JMP and no way to return to the calling program.

When a subroutine finishes, it is necessary to "clean-up" the stack by eliminating or skipping over the subroutine arguments. One way this can be done is by insisting that the subroutine keep the number of arguments as its first stack item. Returns from subroutines would then involve calculating the amount by which to reset the stack pointer, resetting the stack pointer, then restoring the original contents of the register which was used as the copy of the stack pointer. The PDP-11, however, has a much faster and simpler method of performing these tasks. The MARK instruction which is stored on a stack in place of "number of argument" information may be used to automatically perform these "clean-up" chores. (For more information on the MARK instruction refer to Chapter 4.)

7.2.4 PDP-11 Subroutine Advantages

There are several advantages to the PDP-11 subroutine calling procedure.

a. arguments can be quickly passed between the calling program and the subroutine.

b. if the user has no arguments or the arguments are in a general register or on the stack the JSR PC,DST mode can be used so that none of the general purpose registers are taken up for linkage.

c. many JSR's can be executed without the need to provide any saving procedure for the linkage information since all linkage information is automatically pushed onto the stack in sequential order. Returns can simply be made by automatically popping this information from the stack in the opposite order of the JSR's.

Such linkage address bookkeeping is called automatic "nesting" of subroutine calls. This feature enables the programmer to construct fast,

efficient linkages in a simple, flexible manner. It even permits a routine to call itself in those cases where this is meaningful. Other ramifications will appear after we examine the PDP-11 interrupt procedures.

7.3 INTERRUPTS

7.3.1 General Principles

Interrupts are in many respects very similar to subroutine calls. However, they are forced, rather than controlled, transfers of program execution occurring because of some external and program-independent event (such as a stroke on the teleprinter keyboard). Like subroutines, interrupts have linkage information such that a return to the interrupted program can be made. More information is actually necessary for an interrupt transfer than a subroutine transfer because of the random nature of interrupts. The complete machine state of the program immediately prior to the occurrence of the interrupt must be preserved in order to return to the program without any noticeable effects. (i.e. was the previous operation zero or negative, etc.) This information is stored in the Processor Status Word (PS). Upon interrupt, the contents of the Program Counter (PC) (address of next instruction) and the PS are automatically pushed onto the R6 system stack. The effect is the same as if:

```
MOV PS  ,—(SP)          ;Push PS
MOV R7,—(SP)            ;Push PC
```

had been executed.

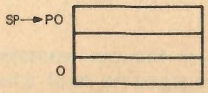
The new contents of the PC and PS are loaded from two preassigned consecutive memory locations which are called an "interrupt vector." The actual locations are chosen by the device interface designer and are located in low memory addresses of Kernel virtual space (see interrupt vector list, Appendix A). The first word contains the interrupt service routine address (the address of the new program sequence) and the second word contains the new PS which will determine the machine status including the operational mode and register set to be used by the interrupt service routine. The contents of the interrupt service vector are set under program control.

After the interrupt service routine has been completed, an RTI (return from interrupt) is performed. The two top words of the stack are automatically "popped" and placed in the PC and PS respectively, thus resuming the interrupted program.

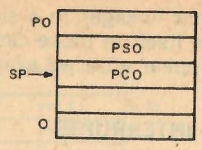
7.3.2 Nesting

Interrupts can be nested in much the same manner that subroutines are nested. In fact, it is possible to nest any arbitrary mixture of subroutines and interrupts without any confusion. By using the RTI and RTS instructions, respectively, the proper returns are automatic.

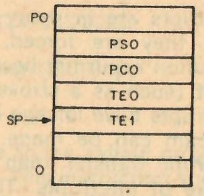
1. Process 0 is running; SP is pointing to location P0.



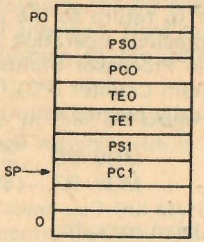
2. Interrupt stops process 0 with PC = PC0, and status = PS0; starts process 1.



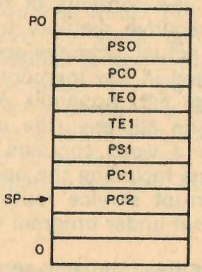
3. Process 1 uses stack for temporary storage (TE0, TE1).



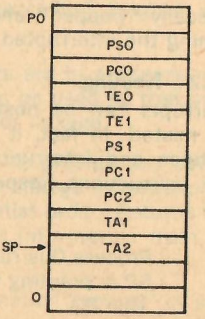
4. Process 1 interrupted with PC = PC1 and status = PS1; process 2 is started



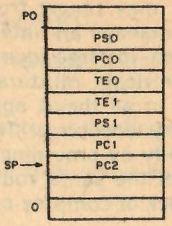
5. Process 2 is running and does a JSR R7,A to Subroutine A with PC = PC2.



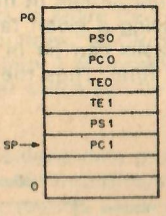
6. Subroutine A is running and uses stack for temporary storage.



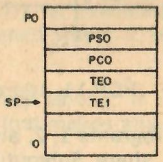
7. Subroutine A releases the temporary storage holding TA1 and TA2.



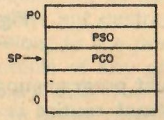
8. Subroutine A returns control to process 2 with an RTS R7,PC is reset to PC2.



9. Process 2 completes with an RTI instruction (dismisses interrupt) PC is reset to PC1 and status is reset to PS1; process 1 resumes.



10. Process 1 releases the temporary storage holding TE0 and TE1.



11. Process 1 completes its operation with an RTI is reset to PC0 and status is reset to PS0.

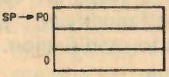


Figure 7-14 Nested Interrupt Service Routines and Subroutines

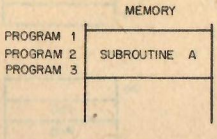
Note that the area of interrupt service programming is intimately involved with the concept of CPU and device priority levels.

7.4 REENTRANCY

Further advantages of stack organization becomes apparent in complex situations which can arise in program systems that are engaged in the concurrent handling of several tasks. Such multi-task program environ-

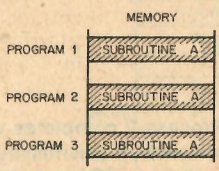
ments may range from relatively simple single-user applications which must manage an intermix of I/O interrupt service and background computation to large complex multi-programming systems which manage a very intricate mixture of executive and multi-user programming situations. In all these applications there is a need for flexibility and time/memory economy. The use of the stack provides this economy and flexibility by providing a method for allowing many tasks to use a single copy of the same routine and a simple, unambiguous method for keeping track of complex program linkages.

The ability to share a single copy of a given program among users or tasks is called reentrancy. Reentrant program routines differ from ordinary subroutines in that it is unnecessary for reentrant routines to finish processing a given task before they can be used by another task. Multiple tasks can be in various stages of completion in the same routine at any time. Thus the following situation may occur:



PDP-11 Approach

Programs 1, 2, and 3 can share Subroutine A.



Conventional Approach

A separate copy of Subroutine A must be provided for each program.

Figure 7-15 Reentrant Routines

The chief programming distinction between a non-shareable routine and a reentrant routine is that the reentrant routine is composed solely of "pure code," i.e., it contains only instructions and constants. Thus, a section of program code is reentrant (shareable) if and only if it is "non self-modifying," that is it contains no information within it that is subject to modification.

Using reentrant routines, control of a given routine may be shared as illustrated in Figure 7-16.

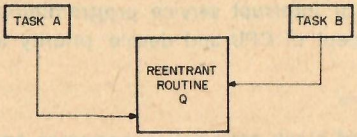


Figure 7-16 Reentrant Routine Sharing

1. Task A has requested processing by Reentrant Routine Q.

2. Task A temporarily relinquishes control (is interrupted) of Reentrant Routine Q before it finishes processing.

3. Task B starts processing in the same copy of Reentrant Routine Q.

4. Task B relinquishes control of Reentrant Routine Q at some point in its processing.

5. Task A regains control of Reentrant Routine Q and resumes processing from where it stopped.

The use of reentrant programming allows many tasks to share frequently used routines such as device interrupt service routines, ASCII-Binary conversion routines, etc. In fact, in a multi-user system it is possible, for instance, to construct a reentrant FORTRAN compiler which can be used as a single copy by many user programs.

As an application of reentrant (shareable) code, consider a data processing program which is interrupted while executing a ASCII-to-Binary subroutine which has been written as a reentrant routine. The same conversion routine is used by the device service routine. When the device servicing is finished, a return from interrupt (RTI) is executed and execution for the processing program is then resumed where it left off inside the same ASCII-to-Binary subroutine.

Shareable routines generally result in great memory saving. It is the hardware implemented stack facility of the PDP-11 that makes shareable or reentrant routines reasonable.

A subroutine may be reentered by a new task before its completion by the previous task as long as the new execution does not destroy any linkage information or intermediate results which belong to the previous programs. This usually amounts to saving the contents of any general purpose registers, to be used and restoring them upon exit. The choice of whether to save and restore this information in the calling program or the subroutine is quite arbitrary and depends on the particular application. For example in controlled transfer situations (i.e. JSR's) a main program which calls a code-conversion utility might save the contents of registers which it needs and restore them after it has regained control, or the code conversion routine might save the contents of registers which it uses and restore them upon its completion. In the case of interrupt service routines this save/restore process must be carried out by the service routine itself since the interrupted program has no warning of an impending interrupt. The advantage of using the stack to save and restore (i.e. "push" and "pop") this information is that it permits a program to isolate its instructions and data and thus maintain its reentrancy.

In the case of a reentrant program which is used in a multi-programming environment it is usually necessary to maintain a separate R6 stack for each user although each such stack would be shared by all the tasks of a given user. For example, if a reentrant FORTRAN compiler is to be shared between many users, each time the user is changed,

R6 would be set to point to a new user's stack area as illustrated in Figure 7-17.

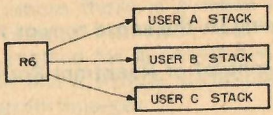


Figure 7-17 Multiple R6 Stack

7.5 POSITION INDEPENDENT CODE—PIC

Most programs are written with some direct references to specific addresses, if only as an offset from an absolute address origin. When it is desired to relocate these programs in memory, it is necessary to change the address references and/or the origin assignments. Such programs are constrained to a specific set of locations. However, the PDP-11 architecture permits programs to be constructed such that they are not constrained to specific locations. These Position Independent programs do not directly reference any absolute locations in memory. Instead all references are "PC-relative" i.e. locations are referenced in terms of offsets from the current location (offsets from the current value of the Program Counter (PC)). When such a program has been translated to machine code it will form a program module which can be loaded anywhere in memory as required.

Position Independent Code is exceedingly valuable for those utility routines which may be disk-resident and are subject to loading in a dynamically changing program environment. The supervisory program may load them anywhere it determines without the need for any relocation parameters since all items remain in the same positions relative to each other (and thus also to the PC).

Linkages to program routines which have been written in position independent code (PIC) must still be absolute in some manner. Since these routines can be located anywhere in memory there must be some fixed or readily locatable linkage addresses to facilitate access to these routines. This linkage address may be a simple pointer located at a fixed address or it may be a complex vector composed of numerous linkage information items.

7.6 CO-ROUTINES

In some situations it happens that two program routines are highly interactive. Using a special case of the JSR instruction i.e., JSR PC, @(R6)+ which exchanges the top element of the Register 6 processor stack and the contents of the Program Counter (PC), two routines may be permitted to swap program control and resume operation where they stopped, when recalled. Such routines are called "co-routines." This control swapping is illustrated in Figure 7-18.

Routine #1 is operating, it then executes:

```
MOV #PC2,—(R6)

JSR PC,@(R6)+
```

with the following results:

1) PC2 is popped from the stack and the SP autoincremented
2) SP is autodecremented and the old PC (i.e. PC1) is pushed
3) control is transferred to the location PC2 (i.e. routine #2)

Routine #2 is operating, it then executes:

```
JSR PC, @(R6)+
```

with the result the PC2 is exchanged for PC1 on the stack and control is transferred back to routine #1.

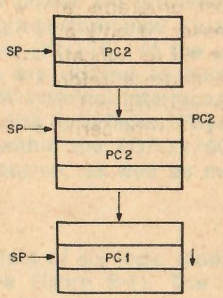


Figure 7-18 Co-Routine Interaction

CHAPTER 8

PROCESSOR AND COMPUTER CHARACTERISTICS

8.1 DESCRIPTION

The PDP-11/60 is designed around basic PDP-11 family architecture. In the processor, the basic PDP-11 instruction set is implemented; extensive microcode allows the additional implementation of the EIS and floating point instructions. Standard memory management allows access to 256K bytes of UNIBUS memory. Space and power are provided for unique PDP-11/60 processor options for a high-performance Floating Point Processor (FP11-E) and User Control Store (KU116). At the computer level, core and MOS UNIBUS memories are utilized as backing store. Peripheral devices range from the DL11-W terminal interface/line clock interface to RK06 and RK05 cartridge disk interfaces. Physical space and adequate power and cooling exist within the PDP-11/60 to support these peripherals. Logic interface expansion, as well as memory expansion, is possible.

8.2 PROCESSOR CHARACTERISTICS

The central processor for the PDP-11/60 consists of six logic modules contained within the backplane assembly (see Figure 8-1). The processor also has space for the optional User Control Store (KU116) and Floating Point Processor (FP11-E). Some additional space is available for single-module UNIBUS interfaces. One such space is used for the Serial Line Interface (DL11-W) which connects to the console terminal.

The processor is a functional unit connected to the UNIBUS and has the features noted below. Some of these features are described in other chapters; the remainder are expanded upon in this chapter.

The PDP-11/60 basic processor features include:

- Basic PDP-11 Instruction Set
- Internal EIS Instructions
- Integral Floating Point Instructions
- Cache Memory
- Memory Management Unit
- Programmable Stack Limit
- Power Fail/Auto Restart
- Four Levels of Priority Interrupt
- UNIBUS Compatibility
- Keypad/Numeric Display Console
- Maintenance Features

New processor options include:

- High Speed Floating Point Processor (FP11-E)

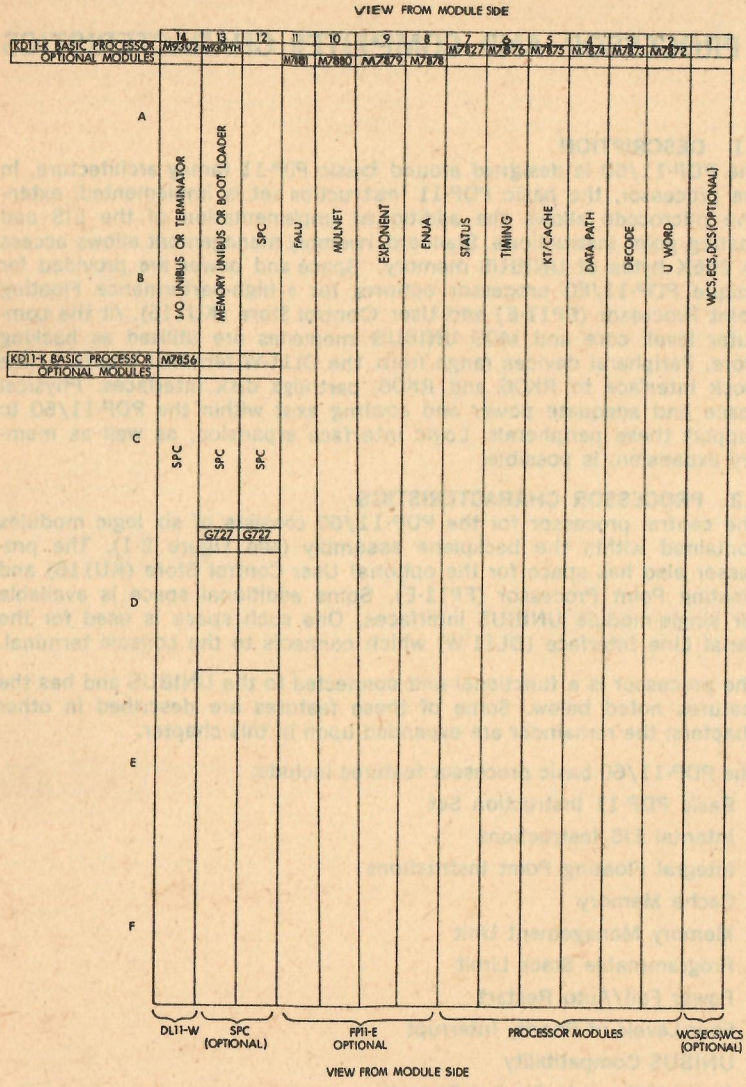


Figure 8-1 PDP-11/60 Processor Backplane

- User Control Store (KU116-AA)
- Extended Control Store (KU116-AB)
- Diagnostic Control Store (KU116-BB)

8.2.1 Central Processor

One set of eight general purpose registers (R0 through R5; Stack Pointer, R6; and Program Counter, R7) is provided within the datapath. Other registers are provided to aid console functions, error log, and the floating point accumulators. Microcode is essential in manipulating the datapath and 2560 control words are provided. The power of these control words is further enhanced with microsubroutine capability and the ability to exchange microcontrol data with macroprogram information. The datapath and microcontrol of the basic machine are noted in Chapter 9.

8.2.2 Extended Instruction Set

The Extended Instruction Set (EIS) provides the capability of performing hardware fixed-point arithmetic and allows direct implementation of multiply, divide, and multiple shifting. A double-precision 32-bit word can be handled. The Extended Instruction Set executes compatibly with the EIS available on the PDP-11/34, PDP-11/35 and PDP-11/40.

8.2.3 Integral Floating Point Unit

The PDP-11/60 contains integral floating point hardware which can execute the full complement of PDP-11 floating point instructions. The instructions are noted in Chapter 10.

8.2.4 Cache Memory

Cache memory simulates a system having a large amount of fast memory. The cache itself uses a small amount of very fast memory, with the larger main memory using slower, more cost-effective MOS or core storage. The cache control logic automatically and dynamically allocates the high-speed cache locations to provide a high probability that required program data will be in the cache. Details are noted in Chapter 6.

8.2.5 Memory Management

Memory management is an advanced memory extension, relocation, and protection feature which will:

- Extend memory space from 56K to 248K bytes
- Allow efficient segmentation of core for multi-user environments
- Provide effective protection of memory segments in multi-user environments

Memory management in the PDP-11/60 is totally compatible with the memory management (KT11-D) option on the PDP-11/34, PDP-11/35 and PDP-11/40.

The machine operates in two modes: Kernel and User. When the machine is in Kernel mode, a program has complete control of the machine; when in User mode the processor is inhibited from executing certain instructions and can be denied direct access to the peripherals or other protected memory locations in the system. This hardware feature can be used to provide complete executive protection in a multi-programming environment. A software operating system can insure that no user

(who is operating in User mode) can cause a failure (crash) of the entire system.

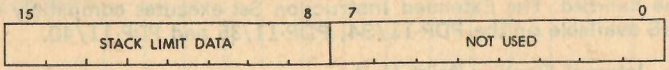
Refer to Chapter 5 for a detailed description of the Memory Management Unit.

8.2.6 Programmable Stack Limit

The stack limit allows program control of the lower limit for permissible stack addresses. This limit may be varied in increments of $(400)_8$ bytes or $(200)_8$ words, up to a maximum address of 177400 (almost the top of a 32K memory).

The normal boundary for stack addresses is 400. The stack limit option allows this lower limit to be raised, providing more address space for interrupt vectors or other data that should not be destroyed by the program.

There is a Stack Limit Register, with the following format:



The Stack Limit Register can be addressed as a word at location 777774, or as a byte at location 777775. The register is accessible to the processor and console, but not to any bus device.

The 8 bits, 15 through 8, contain the stack limit information. These bits are cleared by System Reset, Console Start, or the RESET instruction. The lower 8 bits are not used. Bit 8 corresponds to a value of $(400)_8$ or $(256)_{10}$.

The contents of the Stack Limit Register (SL) are compared to the stack address to determine if a violation has occurred (although memory references that do not alter memory are always allowed). The least significant bit of the register (bit 8) has a value of $(400)_8$. The determination of the violation zones is as follows:

Yellow Zone $=$ (SL) $+$ (340 through 377)$_8$ execute, then trap.
Red Zone $\leqslant$ (SL) $+$ $(337)_8$ abort, then trap to location 4.

The Stack Limit Register contents were zero:

Yellow Zone $=$ 340 through 377
Red Zone $=$ 000 through 337

8.2.7 Power Fail/Auto Restart

Whenever ac power drops below 95 Volts for 115 Volt power (190 Volts for 220 Volt power), the power fail sequence is initiated. The central processor automatically traps to location 24 and the power fail program has 2 milliseconds to save all volatile information (data in registers) and to condition peripherals for power fail.

When power is restored, the processor traps to location 24 and executes the power-up routine to restore the machine to its state prior to power

failure. If MOS memory is present, and its battery backup has failed, the processor, in Panel Lock mode, will alter its power-up sequence to the bootstrap loader.

8.2.8 Priority Interrupt

The PDP-11/60 interrupt system has four priority levels, each of which can handle an almost unlimited number of devices. The priority of the device is a function of the device's physical location—the closer to the processor, the higher its priority on that level.

The priority system makes excellent use of the PDP-11's hardware stacks. When the processor services an interrupt, it first saves important program information on the stack. This information enables the processor to automatically return to the same point in the program and the same conditions, once the current interrupt has been serviced.

The device causing the interrupt(s) provides a direct vector to its own service routine—eliminating the slow and tedious operation of polling all devices to see which one interrupted.

The device also provides status information for its own service routine. Thus, the programmer has the flexibility of assigning a device to a higher priority and its service routine to a lower priority without writing special software.

The system also allows interrupts to be enabled or disabled, through software, during program operation. Such masking allows priorities to change dynamically in response to system conditions.

For example, a real-time program can disable data entry terminals whenever critical analog data is being collected. As soon as the scan is complete, the terminals can be automatically enabled and ready to input data.

8.2.9 UNIBUS Compatibility

All PDP-11/60 system elements—processor, memory, peripherals—plug into a single asynchronous high-speed bus, which transfers all addresses and data. Known as the UNIBUS, this bidirectional bus enables easy interfacing, simplifying the construction of multiprocessor or shared peripheral configurations. Because it is asynchronous, the UNIBUS is compatible with devices that operate over a wide range of speeds. Faster devices or memory can always replace older versions without affecting the system.

Through the UNIBUS, fast devices have easy direct-memory access through cycle-stealing. They require no expensive multiplexers or synchronous DMA hardware. These devices can send, receive, or exchange data without processor intervention and without intermediate buffering in memory.

8.2.10 Keypad/Numeric Display Console

The PDP-11/60 processor provides a direct interface to the user in the KY11-P programmer console. The octal display and keypad make this interface convenient. Through the keypad, direct access is provided to machine registers as well as memory. Control functions allow direct initialization of the standard bootstrap loader, or the optional diagnostic

control store. Details of operation and overall use within the maintenance area are presented in Chapter 11.

8.2.11 Maintenance Features

PDP-11 computers provide a range of maintenance-oriented software and hardware features. At the computer level, system and peripheral diagnostics are used to isolate errors for repair; the central processor is the major tool for diagnostic work. In the PDP-11/60, special attention has been paid to this role and the demand for an ever-operational processor. Extra control and error status registers are provided for the cache and bus operations. An error log can be created at the point of error. Microdiagnostics are available in the Diagnostic Control Store (KU116-BB) and are within the base machine to support console debug.

8.2.12 FP11-E High-Speed Floating Point Processor Option

The FP11-E Floating Point Processor is an optional, asynchronous, parallel processor capable of doing high-speed arithmetic calculations. The FP11-E is logically contained on four hex modules that fit into the processor backplane (see Figure 8-1). A separate power supply is also supplied with the unit.

The FP11-E provides 17 digits of decimal accuracy, does 32-bit single-precision or 64-bit double-precision arithmetic and contains six 64-bit accumulators. Additional information may be found in Chapter 10.

8.2.13 User Control Store, Extended Control Store and Diagnostic Control Store Options

These optional expansions of the control store of the PDP-11/60 processor accommodate different needs. The User Control Store provides a Read And Modify (RAM) expansion of 1024 microwords and is suitable for microprogram development and local store. The Extended Control Store is a vehicle for developed customer or DIGITAL applications in non-alterable Programmable Read Only Memory (PROM). The Diagnostic Control Store allows the application of the microcontrol for diagnostic use within the processor. All these options are discussed further in Chapter 9.

8.3 COMPUTER CHARACTERISTICS

Basic Computer

The basic computer for the PDP-11/60 includes the processor, memory and I/O devices, all connected to the UNIBUS. The processor, discussed in paragraph 8.2, is common to all PDP-11/60s; also common are the M9302 Bus Terminator and the M9301-YH Bootstrap Loader/Terminator. The DL11-W Serial Line/Line Clock Interface is included in the basic PDP-11/60 and is usually used with the console terminal device. Memories can vary the computer model, and are presently core or MOS technology. These units are functionally similar with respect to the UNIBUS. MOS units use Error Correcting Codes (ECC) for reliability; core units use parity. Battery backup is provided with the MOS memory to prevent loss of memory contents due to power failure.

System Computer

The system computer for the PDP-11/60 goes beyond the basic processor/memory/I/O device to include several mass storage devices. Be-

cause these devices and other peripherals are PDP-11/UNIBUS compatible, a number of configurations are possible.

Expansion capabilities are noted for the system computer in terms of power and space.

8.3.1 M9302 UNIBUS Terminator

Besides proper UNIBUS termination, this device contains logic to aid system throughput. Unheeded grants to peripherals are answered and it is not necessary for the processor to time out to continue system activity.

8.3.2 M9301-YH Bootstrap Loader/Self Diagnostic/Terminator

The M9301-YH module, which is included with the PDP-11/60, provides three functions for the computer system.

1. It contains a read-only memory (ROM) that holds diagnostic routines for verifying computer operation.
2. It contains, also in ROM, the several bootstrap loader programs for starting up the system.
3. It provides termination resistors for the UNIBUS.

Diagnostics

The M9301-YH contains diagnostics to check the processor, cache memory and main memory in a GO/NO GO mode. This allows the user to detect the general area of a malfunction. Further localization of the malfunction involves examining certain associated registers. Execution of the diagnostics occurs automatically upon loading, but may be disabled by switches on the M9301-YH. It is also possible to inhibit only the memory modifying tests (cache or memory).

Bootstrap Loaders

The M9301-YH contains independent bootstrap programs that can bootstrap programs into memory from a selected peripheral device. Through front panel control (CONTROL and BOOT switches pressed simultaneously) or following power-up, the computer can directly execute a bootstrap, without the operator having to manually key in the initial program. The bootstrap program for the peripheral device is determined by switches on the M9301. This is useful in remote applications where no operator is present.

The inclusion of a bootstrap loader in non-destructible read-only memory is a strong convenience in system operation. Bootstrap programs do not have to be manually loaded into the computer for system initialization. If the device that is normally booted is down, the user can boot from any other device via the console.

The M9301-YH Bootstrap Loader supports the following devices:

- RK06 Dual-Access Drive
- RK05 Disk Drive
- RH11/RP04, RP05, RP06 Massbus Controller/Disk Drive
- RX01 Floppy Disk Drive
- RS04 Fixed-Head Disk Drive

- PC11 High-Speed Reader and Punch
- TU10 Tape Transport
- TU16 Magtape Transport
- TU56 Dual DECtape Transport
- RP03 Moving Head Disk

8.3.3 DL11-W Serial Line Unit/Line Frequency Clock

The DL11-W serial line unit/line frequency clock provides both a line frequency clock and an asynchronous serial line interface. The line frequency clock allows interrupts at intervals of 16 msec or 20 msec determined by either a 60 Hz or 50 Hz line frequency.

The serial line interface converts serial data to parallel or parallel data to serial to provide communication between the PDP-11/60 and an asynchronous serial line which is connected to a terminal device. The serial line interface has two modes of operation: Current Loop mode (standard on PDP-11/60) and EIA.

8.3.4. UNIBUS Memories

Two characteristics of the PDP-11/60 processor provide flexible memory selection at the system level:

- UNIBUS compatibility allows direct access to a number of present and yet to be developed memory units;
- Cache memory within the processor eliminates absolute dependence upon bus transfer and memory access times in determining instruction execution time.

The PDP-11/60, therefore, will use a number of memory types to meet customer needs. Two such memories are noted below:

MF11-WP Memory

The MF11-WP memory is a read/write, random access, coincident current, magnetic core memory with a maximum cycle time of 1130 nanoseconds and a maximum access time of 670 nanoseconds. It is organized in a 3D, 3-wire planar configuration. Word length is 18 bits (2 bytes of 8-bit data plus parity). The memory consists of 32,768 (32K) words.

Single error parity is associated with each 8-bit byte. The memory can be interleaved within a 64K increment for faster operation. Interleaving causes consecutive bus addresses to be located within alternate 32K memory blocks.

MF11S-KF MOS Memory

The MF11S-KF MOS memory is designed to interface with the PDP-11 UNIBUS. It consists of a control card and from one to four MOS memory storage array cards.

The control card is a hex multilayer board that contains the UNIBUS interface, timing chain, error correction logic, control and status register and refresh circuitry.

The storage array card is a hex multilayer board which has a maximum capacity of 32K x 16. The maximum capacity of 128K x 16 is obtained by using four 32K x 16 array cards.

The array card is internally organized as 16K x 32 to maximize the efficiency of using the error correction logic. An internal bus is used to communicate between the control and array cards.

The error correction logic will detect and correct single bit errors and detect double bit errors. The single error detection and correction will be transparent to the master device accessing the memory.

A double error will be treated in a similar manner to a parity error occurring with an M7850 parity controller.

8.3.5 Basic PDP-11/60 System

The PDP-11/60 disk-based systems are housed in a double-width lowboy cabinet.

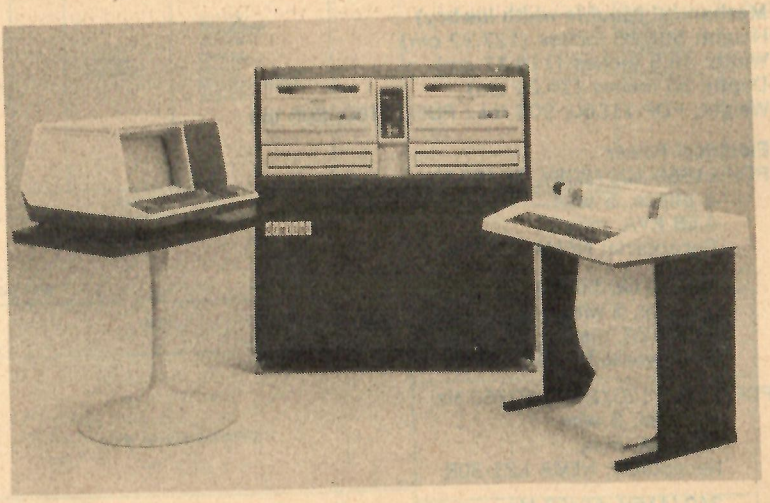


Figure 8-2 PDP-11/60 System with RK06 Disks

In addition to the basic PDP-11/60 computer, the systems have two disk drives, an expansion card cage and power supply (BA11-PE), and a power controller (866). The PDP-11S60 uses the RK06 disks; the PDP-11T60 uses the RK05 disks (see Figure 8-2).

RK06 Disk Characteristics

The RK06 disk drive is a high-performance medium-capacity mass storage device. Using technology similar to the RP04/05/06 drives, the RK06 provides the reliability and features of a high-capacity drive in a medium range, inexpensive subsystem. The RK06 disk cartridge holds approximately 14 million bytes. The system is expandable to more than 111 million bytes, approximately.

RK05 Disk Characteristics

The RK11-D DECpack cartridge disk drive and control is a complete mass storage system, offering an economical solution for large volume, ran-

dom-access data storage. An RK11-D includes a control unit and the first disk drive (RK05).

The system includes a modular mass storage device utilizing removable disk cartridges and a complete easy-to-program control. A disk cartridge holds over 1.2 million words. The DECpack is ideal where a large volume of programs and data is developed and maintained for one or more users. The system is expandable up to 9.6 million words per control (8 disks).

8.3.6 Computer System Specifications

Environment
Operating Temperature: 5° C to 50° C
Relative Humidity: 10% to 90%, non-condensing

Mechanical (double-width lowboy)
Height: 50.125 inches (127.32 cm)
Width: 46.5 inches (118.11 cm)
Depth: 30 inches (76.20 cm)
Weight: PDP-11S60, 900 lbs.; PDP-11T60, 800 lbs.

Electrical Power
PDP-11S60 120/208V, 50/60 Hz
3 phase, 5 wire
4.68 KVA, max.
Receptacle: NEMA L21-30R

240/416V, 50/60 Hz
3 phase, 5 wire
4.68 KVA, max.
Receptacle: NEMA L22-20R

PDP-11T60 120/208V, 50/60 Hz
3 phase, 5 wire
4.2 KVA, max.
Receptacle: NEMA L21-30R

240/416V, 50/60 Hz
3 phase, 5 wire
4.2 KVA, max.
Receptacle: NEMA L22-20R

8.3.7 Expansion Capability

Space
At the system level, the basic computer components, with the logic interface for disk controllers, are arranged as noted. Within the expander card cage, there are four system units of expanison.

dc Power
The PDP-11/60 systems utilize two H7420 base power supplies. Different regulators within these power supplies provide a variety of dc voltages.

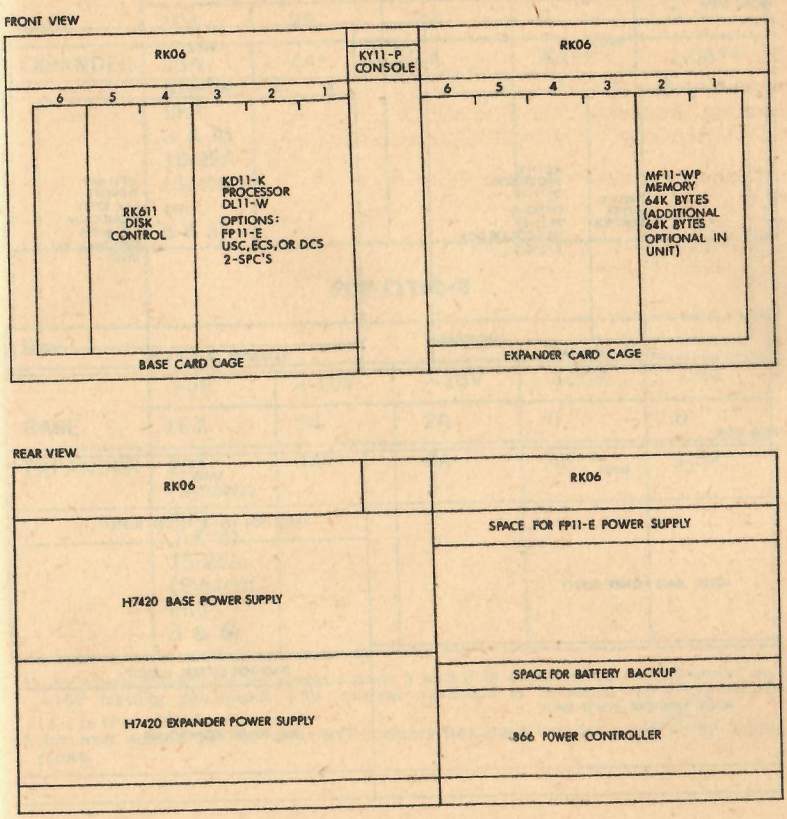


Figure 8-3 PDP-11S60-B Configuration

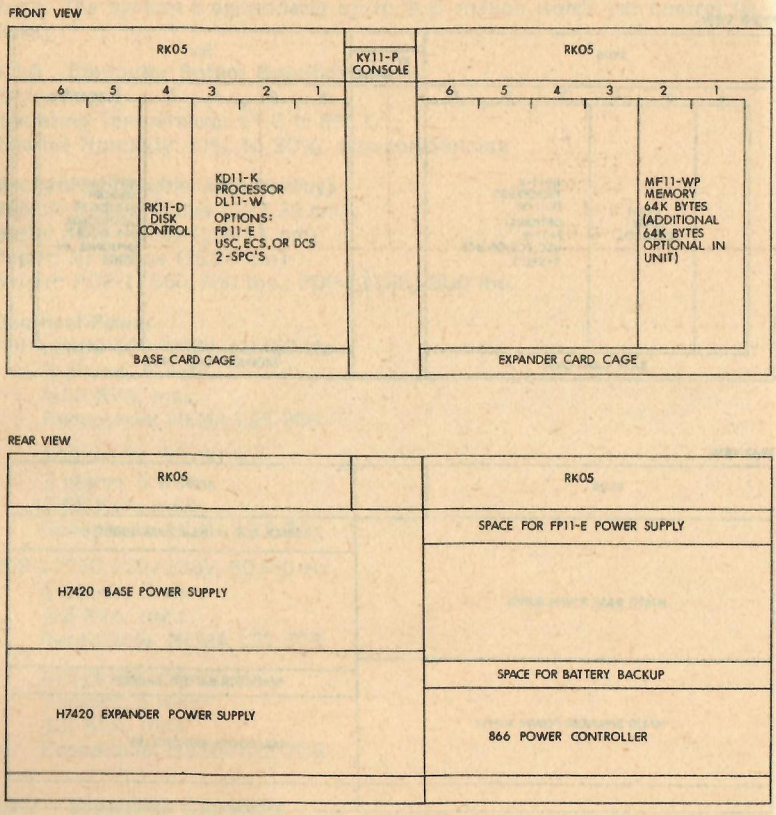


Figure 8-4 PDP-11T60-B Configuration

Table 8.1 Core Expansion Power

PDP-11S60-B

Box	Voltage				
	+5V	+15V	−15V	+20V	−5V
BASE	10A	2A	2A	0	0
EXPANDER	25A (System unit 3 & 4) 15-25A (System unit 5 & 6)	4A*	4A	4A**	1.5A**

PDP-11T60-B

Box	Voltage				
	+5V	+15V	−15V	+20V	−5V
BASE	16A	2A	2A	0	0
EXPANDER	25A (System unit 3 & 4) 15-25A (System unit 5 & 6)	4A*	4A	4A**	1.5A**

* +5V current available for system units 5 and 6 in expander box is dependent on +15V loading. Maximum +5V current available is 15 Amps $\frac{+4 - I_{+15}}{0.4}$ where I_{+15} is the +15V load current.

** Assumes second 32K of MM11-WP memory has been installed in MF11-WP backplane.

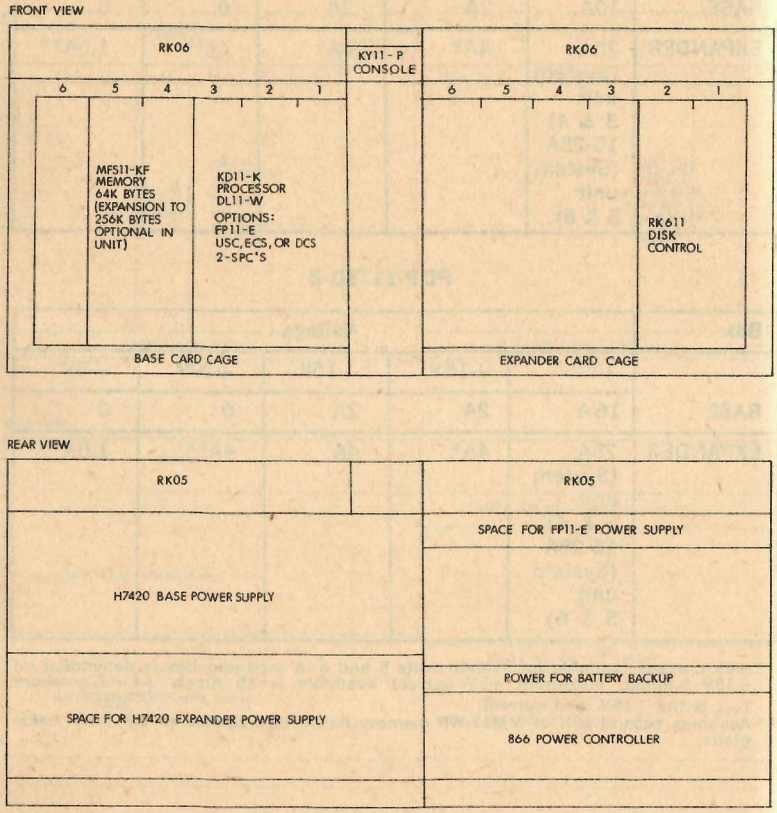


Figure 8-5 PDP-11S60-C Configuration

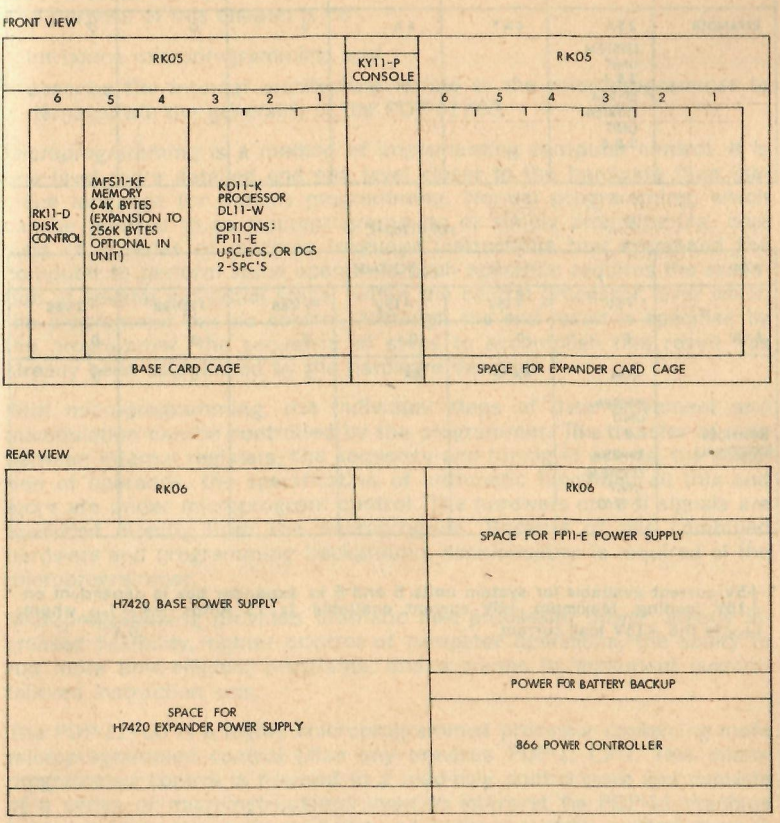


Figure 8-6 PDP-11T60-C Configuration

Table 8.2 MOS Expansion Power

PDP-11S60-C

BOX	VOLTAGE					
	+5V	+15V	-15V	+5 VBB	+12 VBB	-12 VBB
BASE	18A	2A	2A	2.5A	.1A	0.15A
EXPANDER	25A (SYSTEM UNIT 3 & 4 15-25A (SYSTEM UNIT 5 & 6	4A *	4A	0	0	0

PDP-11T60-C

BOX	VOLTAGE					
	+5V	+15V	-15V	+5 VBB	+12 VBB	-12 VBB
BASE	0	0	0	0	0	0
EXPANDER (OPTIONAL)	50A (SYSTEM UNIT 1,2,3 & 4) 15-25A (SYSTEM UNIT 5 & 6)	4A *	4A	0	0	0

* +5V current available for system units 5 and 6 in expander box is dependent on +15V loading. Maximum +5V current available is 15 Amps $\frac{+4 - I_{+15}}{0.4}$ where I_{+15} is the +15V load current.

CHAPTER 9

USER MICROPROGRAMMING

9.1 INTRODUCTION

The purpose of this chapter is to:

- introduce microprogramming, and
- describe the internal architecture visible to the microprogrammer to demonstrate the generality of the PDP-11/60.

Microprogramming is a method of implementing computer control. It is one level more detailed and one level closer to the hardware than machine language (or macro) programming. Normal programming, which can be referred to as macroprogramming or simply programming, consists of a series of machine language instructions that command the computer to perform some operation. Each operation requires the execution of several individual steps, within the central processor, over which the programmer has no control. Although the end result is specified by the programmer, the sequence of steps to accomplish this result has already been established by the hardware designer.

With microprogramming, the individual steps of data movement and manipulation can be controlled by the programmer. The transfer of data between internal registers, the sequence and timing of events, the selection of operands, the specification of arithmetic functions; all this and more are under microprogram control. The hardware control signals are produced directly from the microprogram. Because of this combined hardware and programming background, sophistication is required of the microprogrammer.

Microprogramming provides dramatic new processing power, greatly increased flexibility, tighter control of computer operations, the ability to run more time-efficient programs, and a means to implement custom-tailored instruction sets.

The PDP-11/60 is a highly microprogrammed processor containing more microprogrammed control than any previous PDP-11 CPU. This microprogrammed control is housed in a read-only control store and consists of a series of microinstructions used to interpret the PDP-11 machine language instructions. In addition, an optional writable control store is available.

Microprogramming allows the computer processor to interpret a large variety of instructions and provides a high degree of generality in the PDP-11/60 internal architecture. For example, the data for the integral floating point instructions is treated as a set of four 16-bit words within the basic machine datapaths, while the data for the FP11-E Floating Point Processor (which processes the same floating-point instruction set) is treated as a 64-bit word within its wider datapath.

Microprogramming can broadly be divided into two areas: vendor microprogramming and user microprogramming. Vendor microprogramming

involves the use of DIGITAL-supplied microcode routines for specific applications. DIGITAL will continue to increase the amount of vendor microprogramming in the future. An example of this might be the moving of certain operating system functions from software into microcode. These operating system "assists" will be available as ROM (Read Only Memory) options.

User microprogramming is available to the sophisticated user who is willing to master the internal structure of the PDP-11/60 to write his own microcode for the specific applications desired.

The use of microprogramming options will enhance the performance of the PDP-11/60 since they will be marketed as hardware/firmware/software products with specific microcode routines for the various enhancements.

9.1.1 Microprogram Control

Microprogramming is used extensively in the PDP-11/60 processor as a method of implementing control. The processor has an operational unit called the datapath, which is a collection of registers, adders, shifters, scratchpad storage, and other logical units. The execution of a PDP-11 instruction involves a sequence of transfers of data from one register in the datapath to another; some of these transfers take place directly and some through an adder or other logical circuit. Each step (register-to-register transfer, etc.) is controlled by a microinstruction. A set of such microinstructions is called a microprogram. Microinstructions are held in a special memory, called control store, which is accessed once per processor step.

As well as controlling arithmetic operation of the datapath, microinstructions can also be used to control other necessary operations such as fetching the next instruction, address calculation, and initialization of UNIBUS cycles. Many of these operations can be done in the same cycle, thereby providing a high degree of parallelism. This is of little concern to the macroprogrammer, but is significant to the microprogrammer in exploiting the full potential of the machine.

9.1.2 Dynamic Control Structure

The principal use of microprogramming in the PDP-11/60 is the implementation of the PDP-11 instruction set. However, the dynamic control structure of the processor allows implementation of many other functions. These range from instruction set extensions to emulation of a computer with a different instruction set. A processor is said to have a dynamic control structure if its control section can be altered and enlarged.

Microprogramming is the best control method currently available to implement a dynamic control structure. This is because it is more systematic and flexible than the more conventional method of hardwired control. Microprogramming is also a more systematic method of generating sequences of control signals than special-purpose timing logic or macroprogramming.

9.1.3 Simple Datapath

Figure 9-1 shows a simplified datapath similar to that of the PDP-11/60.

The only combinational component is an Arithmetic Logic Unit (ALU), which has two inputs. The ALU result, or output, is stored in the D register. The other components of this datapath are the B register, a scratchpad (which is a set of high-speed storage registers), and interconnections.

The solid arrows in the figure represent the flow of data within this datapath. Two operands (scratchpad and B register outputs) are presented to the ALU inputs; the ALU performs a specified arithmetic or logical operation on these inputs and sends the result to the input of the D register. After storage in D, the result can be transferred to the scratchpad registers. In turn, data can be written from the scratchpad to the ALU and B register.

To route the flow of data between the components of this datapath, a set of gates, (shown as circles with X's inside, see Figure 9-1), with corresponding control signals, is required. The set of control signals (open arrows) needed is determined by the interconnections between the sequential and combinational units of the datapath.

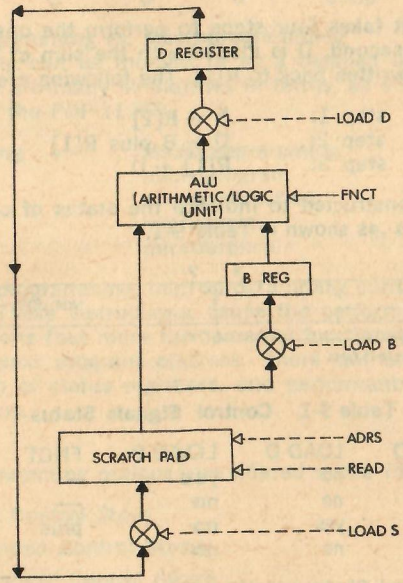


Figure 9-1 Sample Datapath (not PDP-11/60)

For this datapath, the following control signals are needed.

1. ADRS—To specify a location in the scratchpad.
2. READ—To specify reading from the scratchpad registers. WRITE is the negation of READ.

3. LOAD B—To move data from the scratchpad in B.
4. FNCT—To select the ALU function.
5. LOAD D—To move the ALU result in D.

Microprogramming allows control of each of these signals to produce a wide variety of operations.

9.1.4 Example of Microprogramming

A microprogram will be generated to control this datapath for a PDP-11 instruction. An initial condition will be that the 8 PDP-11 general registers (R0 through R7) will be stored in the first 8 locations of the register scratchpad. To perform the PDP-11 operation:

ADD R2, R1

The second and third locations in the scratchpad must be added, and the result stored in the second location (R1). Symbolically, this is represented as:

R1 ← R2 plus R1

This means that R1 is replaced by the sum of R2 and R1.

In this example, it takes four steps to perform the operation. First, R2 is loaded into B; second, D is loaded with the sum of B and R[1]; and last, the result is written back to R[1]. The following are the basic steps:

step 1: B ← R[2]
step 2: D ← B plus R[1]
step 3: R[1] ← D

A table can be constructed to indicate the status of control signals in each of these steps ,as shown in Table 9-1.

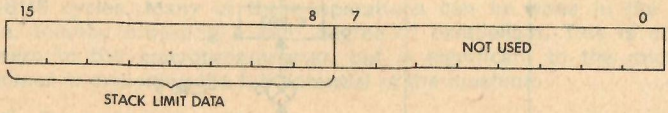


Table 9-1 Control Signals Status

TIME	READ	LOAD D	LOAD B	FNCT	ADRS
step 1	R	no	yes	—	R2
step 2	R	no	no	—	R1
step 3	R	yes	no	plus	R1
step 4	W	no	no	—	R1

By assigning 1's and 0's to specify each operation, it is possible to implement this control with 12 bits (see Figure 9-2). The ALU is allocated 5 bits (FNCT) to allow for a variety of operations; the scratchpad is assumed to have 16 locations, requiring 4 bits (ADRS); and READ, LOAD D, and LOAD B are each one bit.

This microinstruction format, or microword, is divided into fields. Each field comprises the bits which are used to control a particular signal or function.

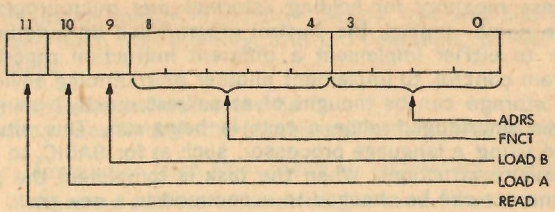


Figure 9-2 Microinstruction Fields

Using the control signal status (Table 9-1) and the microinstruction fields (Figure 9-2), a microprogram to perform the PDP-11 instruction ADD R2, R1 can be coded as:

	READ	D	B	FNCT	ADRS
step 1	1	0	0	0000	0010
step 2	1	0	1	0000	0001
step 3	1	1	0	0001	0001
step 4	0	0	0	0000	0001

9.1.5 Microprogramming vs. Macroprogramming

Microprogramming can be thought of as a detailed level of macroprogramming. There are many similarities in terms, as shown by this comparison chart for the PDP-11/60.

Macroprogramming	Microprogramming
program	microprogram
instruction	microinstruction
machine cycle	microcycle
branch	microbranch

Similar to macroprogramming, microprogramming consists of a sequence of instructions. These instructions cause the performance of a smaller number of functions (but more fundamental functions), select opernads, and select the next program address. There can also be conditional branching, set-up of status registers, and performance of other miscellaneous operations.

9.2 OPTIONS

Three microprogramming options are offered to the PDP-11/60 user:

UCS	User Control Store
ECS	Extended Control Store
DCS	Diagnostic Control Store

These options allow the user to custom-tailor microprograms for his application. The options consist of hardware modules that mount in a dedicated space within the processor backplane. Note that only one of these options may be installed in the computer at the same time. The UCS option includes software tools for program development.

9.2.1 User Control Store

The User Control Store option provides 1,024 forty-eight bit words of

random-access memory for holding (storing) user microprograms and data. As the name implies, the control program can be modified easily by the user to either implement a different instruction repertoire, or, under program control, to implement another programming environment. The control storage can be thought of as a "read mostly" memory that would remain unchanged while a task is being run. This situation is analogous to using a language processor, such as for BASIC, to interpret the higher-level instructions. When the task is completed, the programming environment can be changed to accommodate a new task.

The User Control Store option provides flexibility and control, and is a means for developing special microprogramming for repetitive applications.

9.2.2 Extended Control Store

The Extended Control Store option provides 1,536 forty-eight bit words of read-only memory for a microprogram. In most cases, this will be the result of development work by DIGITAL to provide high performance systems with tailored microcode. With read-only memory, there is no loss of microprogram either through inadvertent program modification, or by power failure.

9.2.3 Diagnostic Control Store

The Diagnostic Control Store option is a hardware aid using microcode analysis of computer operations. It provides a read-only memory that allows isolation of many central processor faults within seconds. The malfunctioning module can then be replaced, and the system can be quickly back in operation. See paragraph 11.4 for more details.

9.2.4 Applications

By design, the PDP-11 is a general purpose computer. Writable Control Store enables the user to tailor the PDP-11/60 to special purpose needs. Such tailoring can be classified hierarchically as follows:

LEVEL 0—Instruction Set Extensions

Some special functions such as block move, decimal arithmetic, and translate are included. Their definition should conform to PDP-11 instruction format and style.

LEVEL 1—Application Kernels

Most applications and system programs have sections which are much more frequently executed than others; 10% of the code is normally executed 90% of the time. Once these critical sections have been identified, reprogramming them at the macro level for faster execution will improve overall execution. The process can be carried a step further; kernels within the critical sections can be microprogrammed for even better throughput. Examples include the Fast Fourier Transform, an operating system's memory allocation routine, and cyclic redundancy check (CRC) calculations.

LEVEL 2—Emulation

The interpretive execution of an instruction set by software is generally called simulation. When this interpretation is done by hardware, it is called emulation. Microprogramming provides a way of inexpensively emulating several different instruction sets on one hardware design. The

tasks of emulation include instruction decode, operand address calculation, operand fetch from memory, and I/O operation, as well as instruction execution.

Analyzing Performance Improvements
An instruction goes through the following phases during its execution:

I-phase	Instruction fetch from memory, Instruction decode
O-phase	Operand address calculation, Operand fetch from memory
E-phase	Execution of the instruction

Each phase takes one or more machine cycles. The total execution time, assuming no overlapping of phases, is the sum of these cycles. Each phase is a candidate for elimination or cycle-reduction with resulting gains in performance.

9.3 PDP-11/60 IMPLEMENTATION

There are many ways in which microprogramming could be achieved. This section will deal only with microprogramming as implemented in the PDP-11/60. Specific details of operation will be presented.

The PDP-11/60 microprogrammable computer system has the following features:

- extra address space in the Control Store
- flexible use of Control Store as Local Store
- means to load and check Control Store
- datapath generality
- microcontrol generality
- microprogrammability (registers, subroutines, constants, intraprocessor control)
- macroprogram access to microcontrol (XFC instruction, trap, default, microbreak)

9.3.1 Control Store Address Space

Enlarging the control section of a microprogrammed computer is relatively straightforward; more control store can be added. If the control store is implemented in random-access memory (RAM), and therefore writable, changing the Control Store becomes even more straightforward by altering the control section. Both enlarging and altering the Control Store are possible within the PDP-11/60 via the UCS/ECS options.

Map of Control Store Address Space
The PDP-11/60 Control Store contains microcode address space which is defined by a 12-bit address. The address space is divided into 8 pages with 512 words allocated to each page. The high order three bits of the twelve-bit address specify page 0 through 7. The remaining nine bits specify one of 512 addresses within the designated page. The lower 2.5K is reserved for the base machine while the upper address space is reserved for UCS/ECS routines. (See Figure 9-3)

9.3.2 Local Store Flexibility

The availability of data storage local to a processor, i.e., not accessed

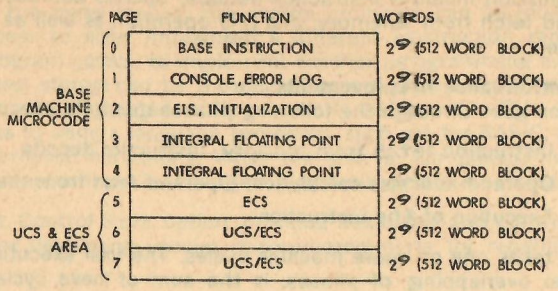


Figure 9-3 Microcode Address Space

through a main, general-purpose memory bus, can have a significant effect on system speed. Such local store is usually implemented in some special technology which has low capacity but high performance.

The User Control Store has been structured so that the 48-bit words can be read and written as 16-bit data words. In addition to conventional writable control store hardware, there is logic to realize a LSAR (local store address register) and LSDR (local store data register).

Thus the microprogrammer has fast local store available. This storage is block-oriented; there is a three-cycle overhead to start a block read (or block write) and then the words are read (or written) at the rate of one per microcycle.

The microprogram can be logically partitioned into two sections:

- control store—48-bit control words
- local store—16-bit data words (3 per microword)

A common partitioning would be 512 words of control store and 1536 words of local store.

Local store provides the microprogrammer with a small, but high-speed memory for storage of often used constants, buffering of data, and a fast-access private scratchpad memory.

9.3.3 Loading/Checking User Control Store

The UCS array is loaded by placing the array address in the WCSAR (777542) and moving the desired data to the WCSDR (777544). Data is read by moving from the WCSDR. These operations are at UNIBUS addresses and use PDP-11 macro instructions.

Parity is generated on each 16-bit data word loaded into the array. Parity is checked across the entire 48-bit word when reading the array, even if only 16 bits are read. Parity errors force a jam-up and trap to location 114. Parity errors are indicated in the WCSAR (777540).

The UCS array resides in pages 6 and 7 of the control store at addresses 6000-7777.

9.3.4 Macro Datapath

Figure 9-4 is a simplified block diagram of the macro datapath in the PDP-11/60. The major functional blocks will be discussed.

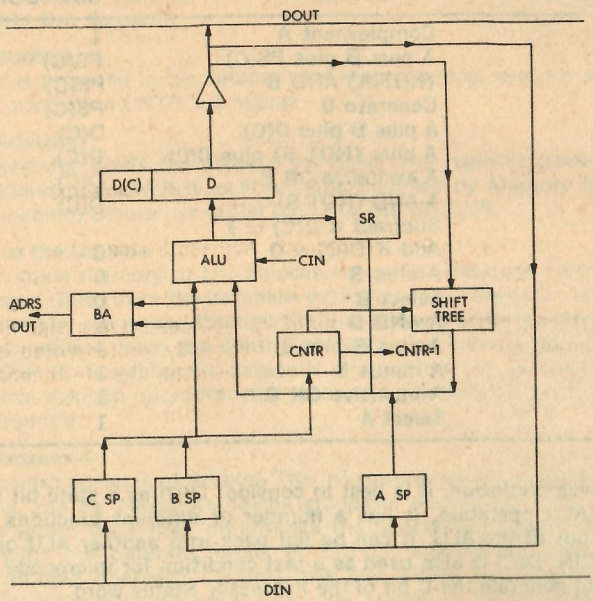


Figure 9-4 Simplified Block Diagram of PDP-11/60 Datapath

ALU (Arithmetic Logic Unit)

The ALU is the major data manipulative element within the datapath. It is a powerful device that accepts two 16-bit operands, plus a one bit operand, and produces a 16-bit output with carry as a function of a 4-bit operation code and the operands. Table 9-2 shows the possible operations. The 16-bit operands are noted as A and B, and a 1-bit operand noted as CIN. CIN is the output of the CIN MUX and can assume a number of values for PDP-11 operations; two such values are PS(C), and D(C).

PS(C) is the C bit of the Processor Status word; the base machine uses it as the D(C) source for some PDP-11 instructions in which the C bit of the PSW does not change. D(C) is the carry bit of the D Register bit generated by the previous CLK D specification. This allows you to save, or recycle, a D(C) value from a previous ALU result.

After an ALU operation is complete, the 16-bit result can be clocked into the D register. If the D register is clocked, one bit from the ALU result may be used for D(C). The bit which becomes D(C) can be the actual carry, or overflow bit of the ALU; hence the term carry-out is used. However, the overflow is not the only possible source for D(C).

Table 9-2 ALU Control Field Encoding

OCTAL VALUE OF ALU FNCT	FUNCTION	CIN SOURCE
0	Complement A	1
1	A plus B plus PS(C)	PS(C)
2	(NOT A) AND B	PS(C)
3	Generate 0	PS(C)
4	A plus B plus D(C)	D(C)
5	A plus (NOT B) plus D(C)	D(C)
6	A exclusive OR B	D(C)
7	A AND (NOT B)	D(C)
10	Subtract if D(C) = 1	
	Add if D(C) = 0	0
11	A plus B	0
12	Select B	0
13	A AND B	0
14	A plus B plus 1	1
15	A minus B	1
16	A inclusive OR B	1
17	Select A	1

To minimize confusion, it is best to consider D(C) as a state bit retained from an ALU operation. It has a number of different functions. As the carry output of the ALU, it can be fed back into another ALU operation through CIN. D(C) is also used as a test condition for microcode branching, and to generate the C bit of the Processor Status word.

ASP (A Scratchpad)
The A Scratchpad consists of 32 16-bit high-speed registers, most of which have dedicated functions within the microprocessor. Eight of these registers are used to hold the contents of the general registers R0 through R7. Complete contents are noted in Chapter 11 under addresses for the MED instruction.

BSP (B Scratchpad)
The B Scratchpad is similar to the A Scratchpad. The 8 general registers are duplicated here to allow concurrent access to these registers. Other registers also exist and are set aside for UCS users. These registers are not accessed or altered by basic PDP-11 execution.

CSP (C Scratchpad)
The C Scratchpad consists of 16 16-bit registers that hold base constants for programs, and data from other parts of the computer (e.g., cache memory or UNIBUS memory).

D Register (Data Register)
The D Register can be loaded with the 16-bit output of the ALU plus the carry bit D(C).

SR (Shift Register)
The SR is a 16-bit register that can be loaded from the ALU. It can be used as a temporary holding register or for shifting operations.

Shift Tree
The Shift Tree performs various operations on data from the D register, such as left shift, multiple right shifts, sign extend, and byte swap. The Shift Tree is also used for shift and test operations on control data brought into the datapath.

CNTR (Counter)
A Counter is provided in the datapath for loop counting, and for a microbranch of control on CNTR = 1 signal.

BA (Bus Address)
The BA Register holds the address for data. The register contains the virtual address of a UNIBUS location. It is modified by Memory Management (if enabled) before use in the cache or the UNIBUS.

Interface to the Outside
Data from main memory or I/O devices enters the datapath through the C Scratchpad. Data from the datapath exists via the D register. Addresses to main memory are routed through the Bus Address (BA) register. Microcontrol information from the control store can be entered as constants through CSP (C Scratchpad); datapath information can be used by the microcontrol through microbranches upon the contents of the D Register and the Counter.

9.3.5 Microword

The microword consists of 48 bits. The microword, when used as a microinstruction, can be viewed as having groups of related bits, called fields. The code within each field determines the operations to be performed. Just as with the different types of PDP-11 instructions (e.g., single operand, double operand, control), the bits within the instruction are interpreted differently depending on the codes in certain fields. See Figure 9-5. Note the character "U" is used to represent Micro (Greek letter μ) in the PDP-11/60.

The PDP-11/60 employs what is known as a "horizontal" microword. That means that a majority of the bits in the microword are directly used to generate some signal within the computer. Some of the fields are encoded, meaning that the value represented by the bits in that field must be decoded before control signals are generated. Some fields have multiple use, depending upon selection by other bits of the microword.

The term horizontal also implies a significant degree of parallelism, or simultaneous control of different functions within the datapath. One microinstruction can, in some circumstances, be much more powerful than one macro-level instruction.

UWORD Functions in a Normal Transfer
When the microword specifies normal data transfer, the following fields are used in the microword.

FIELD	FUNCTION
A OPERAND	Selects the source for the A operand: A scratchpad, shift register, or shift tree.
B OPERAND	Selects the source for the B operand: B scratchpad, C scratchpad (arithmetic constants), or determined from the source or destination field of the macroinstruction.

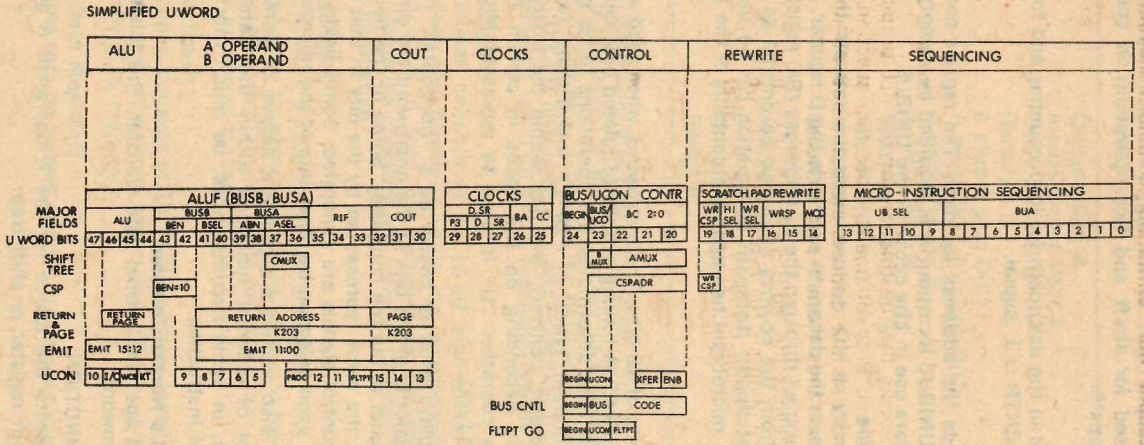


Figure 9-5 Simplified and Detail Representation of the PDP-11/60 Microword

C OUT	Selects the source for the 1-bit C operand; (1 of 8 choices).
ALU	Selects the arithmetic or logical function to be performed. (See Table 9-2)
CLOCKS	Selects when information will be loaded (within the microcycle) and whether the output of the ALU will be transferred to D. Also determines the loading of the SR and BA registers.
CONTROL	Controls activity within the microprocessor and the UNIBUS. An internal microcontrol mechanism, UCON, services low priority transfers of error and control data.
REWRITE	Selects the destination for the ALU output within the A and B scratchpads. Note that selection, without rewrite, is the address mechanism for obtaining A and B operands from the scratchpads.
SEQUENCING	Selects the address for the next microinstruction. Depending on the rest of the microword, this field selects conditional branch inputs to alter next addresses in the microcode.

All these operations are specified in a single microword, such that a high degree of parallelism exists.

9.3.6 Microprogrammability

A general microcode capability requires that the datapath and microcode provide certain features. The salient features of PDP-11/60 microprogrammability are:

- General Registers
- Micro Subroutine Capability
- Constant and Address Generation
- Intraprocessor Control

For control of operational functions within the above features, basic capability of exchange of micro and macro data exists. The microcode can branch its control or count operations from macrodata. The microcode can also provide constants or addresses for macroprogram execution.

High Speed Registers

There are 80 16-bit registers in the datapath. Several of these are reserved for the UCS microprogrammer.

Subroutine and Branching

Because there is no incremental program counter at the microprogramming level, each microinstruction specifies the address of the next microinstruction. Therefore, there is no requirement that microinstructions execute sequentially according to their storage address.

Moreover, each microinstruction can also specify a branch condition to be tested before the next microinstruction is fetched. The result of the test can cause a different microinstruction to be fetched. Information for

such branches comes from a number of sources; some of them from the macro datapath D Register or Counter.

The microword can also specify the return address when exiting from a microprogram subroutine. The programming technique of subroutines is available to the microprogrammer. This usually results in smaller microprograms, more systematic microprogramming, and more easily shared microroutines.

Emit Constant
Constants are frequently needed in microprogramming; for example, the constant 2 for incrementing word addresses or the specification of a UNIBUS address.

The PDP-11/60 allows the convenient introduction of a 16-bit constant directly into the microprogram. One of the fields of the microword has a dual function; in emit mode, the 16 bits are placed in the data flow; in control mode, the bits are interpreted as microcontrol as usual.

Intraprocessor Control
Within the PDP-11/60 processor, there are several distinct functions. Some of these functions, such as datapath or memory management control, must occur at high speed for PDP-11 instruction execution. Other functions, such as Error Log or read/write access to internal processor registers, can occur at slower speed. In these instances, a secondary microcontrol mechanism, called UCON (microcontrol) is utilized. Specific bits in the microword (BEGIN, UCON, XFER, END) cause the reinterpretation of other control bits to enable the secondary functions.

9.3.7 Macro Access to Microcontrol

The most direct program access to microcontrol is through the XFC (Extended Function Code) noted in Chapter 4. In addition, entry into the User Control Store occurs upon the detection of an odd address incoming PC (Program Counter) in a macro trap; and if a reserved instruction code is utilized. These entries occur only if a User Control Store option is present and enabled within the PDP-11/60 processor. A diagnostic entry is also provided in conjunction with the microbreak feature noted in Chapter 12.

This chapter has presented a simplified explanation and description of the microprogramming capabilities within the PDP-11/60. More extensive and detailed information is provided in the documentation associated with the PDP-11/60 User Control Store.

CHAPTER 10

FLOATING POINT PROCESSOR

10.1 INTRODUCTION

The PDP-11/60 contains an integral Floating Point Instruction set supported by microcode in the base machine, and by the optional FP11-E Floating Point Processor. This optional unit fits into the KD11-K processor backplane and provides a high performance execution of the Floating Point Instruction set. (See Instructional Timing in Appendix B.) Both units provide significant improvement in execution over software subroutine implementation of floating point.

This chapter discusses the optional FP11-E Floating Point Processor. Format and instruction information are the same for the integral floating point with the exceptions already noted in Section 4.9. The sequence of operation differs as the FP11-E Floating Point Processor can operate in parallel with the base machine.

The features of the FP11-E unit are:

- 17 digit accuracy
- Overlapped operation with the central processor
- High speed
- Single and double precision (32 or 64 bit) floating point modes
- Flexible addressing modes
- Six 64-bit floating point accumulators
- Error recovery aids

10.2 OPERATION

The Floating Point instruction set is an integral part of the Central Processor. It operates using similar address modes, and the same memory management facilities provided by the Memory Management Option, as the Central Processor. Floating Point Processor (FPP) instructions can reference the floating point accumulators, the Central Processor's general registers, or any location in memory.

When, in the course of a program, an FPP Instruction is fetched from memory, the FPP will execute instruction in parallel with the CPU continuing with its instruction sequence. The CPU is delayed a very short period of time during the FPP Instruction's Fetch operation, and then is free to proceed independently of the FPP. The interaction between the two processors is automatic, and a program can take full advantage of the parallel operation of the two processors by intermixing Floating Point Processor and Central Processor instructions.

Interaction between Floating Point and Central Processor instructions is automatically taken care of by the hardware. When an FPP Instruction is encountered in a program, the machine first initiates Floating Point handshaking and calculates the address of the operand. It then checks

the status of the Floating Point Processor. If the FPP is "busy," the CPU will wait until it is "done" before continuing execution of the program. As an example, consider the following sequence of instructions:

```
        LDD(R3)+,AC3    ;Pick up constant operand and place it
                        in AC3
ADDLP:  LDD(R3)+, AC0   ;Load AC0 with next value in table
        MUL AC3,AC0     ;and multiply by constant in AC3
        ADDD AC0,AC1    ;and add the result into AC1
        SOB R5,ADDLP    ;check to see whether done
        STCDI AC1@R4    ;done, convert double to integer and
                        store
```

In the above example, the Floating Point Processor will execute the first three instructions. After the "ADDD" is fetched into the FPP, the CPU will execute the "SOB", calculate the effective address of the STCDI instruction, and then wait for the FPP to be "done" with the "ADDD" before continuing past the STCDI instruction.

As can be seen from this example, autoincrement and autodecrement addressing automatically adds or subtracts the correct amount to the contents of the register, depending on the modes represented by the instruction.

10.3 ARCHITECTURE

The Floating Point Processor contains scratch registers and six general purpose accumulators (AC0-AC5).

Each accumulator is interpreted to be 32 or 64 bits long depending on the instruction and the status of the Floating Point Processor. For 32-bit instruction only the left-most 32 bits are used, while the remaining 32 bits remain unaffected.

The six Floating Point Accumulators are used in numeric calculations and interaccumulator data transfers; the first four (AC0-AC3) are also used for all data transfers between the FPP and the General Registers or Memory.

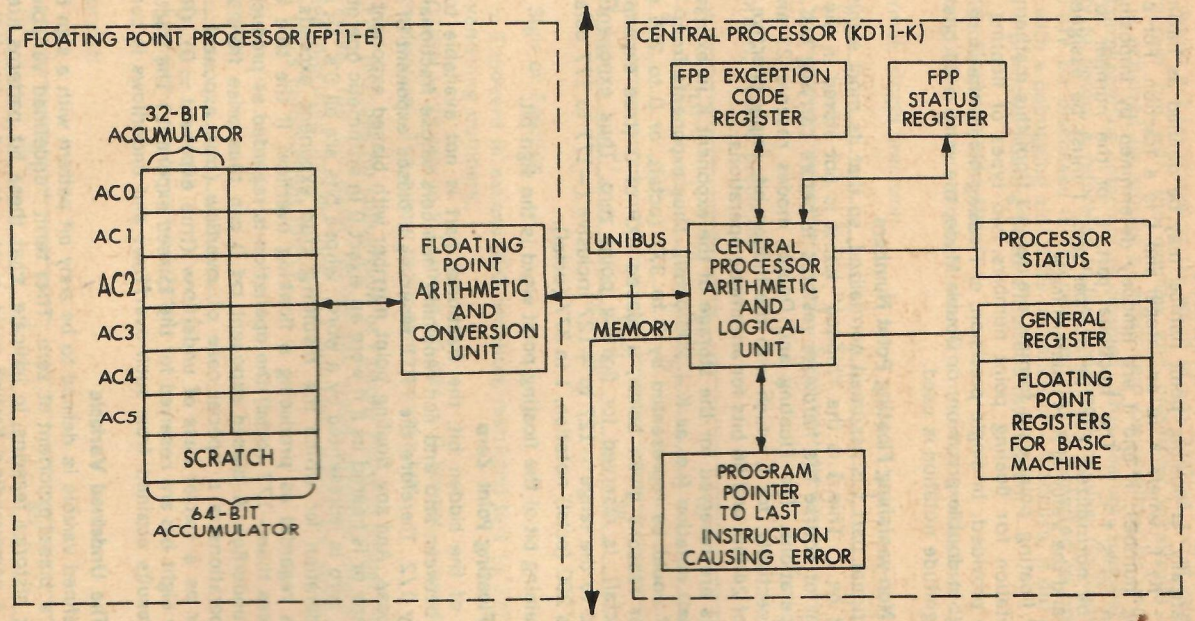


Figure 10-1 Floating Point Processor and Central Processor of the PDP-11/60.

10.4 FLOATING POINT DATA FORMATS

Mathematically, a floating point number may be defined as having the form (2**K)*f, where K is an integer and f is a fraction. For a non-vanishing number, K and f are uniquely determined by imposing the condition $\frac{1}{2} \leq f < 1$. The fractional part, f, of the number is then said to be normalized. For the number zero, f must be assigned the value 0, and the value of K is indeterminate.

The FPP floating point data formats are derived from this mathematical representation for floating point numbers. Two types of floating point data are provided. In single precision, or Floating Mode, the word is 32 bits long. In double precision, or Double Mode, the word is 64 bits long. Sign magnitude notation is used.

10.4.1. Non-vanishing Floating Point Numbers

The fractional part f is assumed normalized, so that its most significant bit must be 1. This 1 is the "hidden" bit: it is not stored in the data word, but of course the hardware restores it before carrying out arithmetic operations. The Floating and Double modes reserve 23 and 55 bits, respectively, for f, which with the hidden bit, imply effective word lengths of 24 bits and 56 bits for arithmetic operations.

Eight bits are reserved for the storage of the exponent K in excess 128 (200 octal) notation (i.e. as K + 200 octal). Thus exponents from −128 to +127 could be represented by 0 to 377 (octal), or 0 to 255 (decimal). For reasons given below, a biased EXP of 0 (true exponent of −200 octal), is reserved for floating point zero. Thus exponents are restricted to the range −127 to +127 inclusive (−177 to 177 octal) or, in excess 200 (octal) notation, 1 to 377 (octal).

The remaining bit of the floating point word is the sign bit.

10.4.2. Floating Point Zero

Because of the hidden bit, the fractional part is not available to distinguish between zero and non-vanishing numbers whose fractional part is exactly 1/2. Therefore the FP11 reserves a biased exponent of 0 for this purpose. And any floating point number with biased exponent of 0 either traps or is treated as if it were an exact 0 in arithmetic operations. An exact zero is represented by a word, whose bits are all 0's. An arithmetic operation for which the resulting true exponent exceeds 177 (octal) is regarded as producing a floating overflow; if the true exponent is less than −177 (octal) the operation is regarded as producing a floating underflow. A biased exponent of 0 can thus arise from arithmetic operations as a special case of overflow (true exponent = 400 octal), or as a special case of underflow (true exponent = 0). (Recall that only eight bits are reserved for the biased exponent.) The fractional part of results obtained from such overflows and underflows is correct.

10.4.3. The Undefined Variable

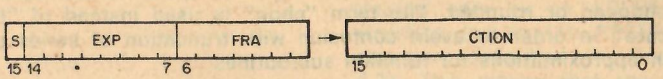
The undefined variable is defined to be any bit pattern with a sign bit of one and a biased exponent of zero. The term "undefined variable" is used, for historical reasons, to indicate that these bit patterns are not assigned a corresponding floating point arithmetic value. Note that the undefined variable is frequently referred to as "−0" elsewhere in this chapter.

A design objective of the FP11-Ewas to assure that the undefined variable would not be stored as the result of any floating point operation in a program run with the overflow and underflow interrupts disabled. This is achieved by storing an exact zero on overflow or underflow, if the corresponding interrupt is disabled. This feature together with an ability to detect a reference to the undefined variable (implemented by the FIUV bit discussed in the next section) is intended to provide the user with a debugging aid: if the presence of −0 occurs, it did not result from a previous floating point arithmetic instruction.

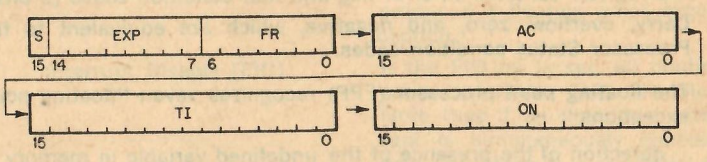
10.4.4. Floating Point Data

Floating point data is stored in words of memory as illustrated below.

F Format, single precision



D Format, double precision



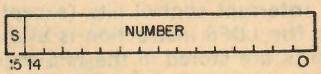
S = Sign of Fraction

EXP = Exponent in excess 200 notation, restricted to 1 to 377 octal for non-vanishing numbers.

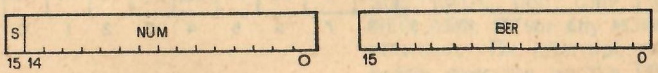
FRACTION = 23 bits in F Format, 55 bits in D Format, + one hidden bit (normalization). The binary radix point is to the left.

The FPP provides for conversion of Floating Point to Integer Format and vice-versa. The processor recognizes single precision integer (I) and double precision integer long (L) numbers, which are stored in standard two's complement form:

I Format:



L Format:



where

S = Sign of Number

NUMBER = 15 bits in I Format, 31 bits in L Format.

10.5 FLOATING POINT UNIT STATUS REGISTER (FPS Register)

This register provides (1) mode and interrupt control for the floating point unit, and (2) conditions resulting from the execution of the previous instruction.

Four bits of the FPS register control the modes of operation:

Single/Double: Floating point numbers can be either single or double precision.

Long/Short: Integer numbers can be 16 bits or 32 bits.

Chop/Round: The result of a floating point operation can be either chopped or rounded. The term "chop" is used instead of "truncate" in order to avoid confusion with truncation of series used in approximations for function subroutines.

Normal/Maintenance: a special maintenance mode is available.

The FPS register contains an error flag and four condition codes (5 bits):

Carry, overflow, zero, and negative, which are equivalent to the Processor Status condition codes.

The floating point processor (FPP) recognizes seven "floating point exceptions":

detection of the presence of the undefined variable in memory
floating overflow
floating underflow
failure of floating to integer conversion
maintenance trap
attempt to divide by zero
illegal floating OP code

For the first five of these exceptions, bits in the FPS register are available to individually enable or disable interrupts. An interrupt on the occurrence of either of the last two exceptions can be disabled only by setting a bit which disables interrupts on all seven of the exceptions, as a group.

Of the fourteen bits described above, five are set by the FPP as part of the output of a floating point instruction: the error flag and condition codes. Any of the mode and interrupt control bits (except the FMM bit) may be set by the user; the LDFS instruction is available for this purpose. These fourteen bits are stored in the FPS register as follows:

FER	FID	UNUSED	FIUV	FIU	FIV	FIC	FD	FL	FT	FMM	FN	FZ	FV	FC
15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0

BIT	NAME	DESCRIPTION
15	Floating Error (FER)	The FER bit is set by the FPP if 1. division by zero occurs 2. illegal OP code occurs 3. any one of the remaining occurs and the corresponding interrupt is enabled. Note that the above action is independent of whether the FID bit (next item) is set or clear. Note also that the FPP never resets the FER bit. Once the FER bit is set by the FPP, it can be cleared only by an LDFPS instruction (or by the RESET instruction described in Section 4.7). This means that the FER bit is up to date only if the most recent floating point instruction produced a floating point exception.
14	Interrupt Disable (FID)	If the FID bit is set, all floating point interrupts are disabled. Note that if an individual interrupt is simultaneously enabled, only the interrupt is inhibited; all other actions associated with the individual interrupt enabled take place.

NOTES

1. The FID bit is primarily a maintenance feature. It should normally be clear. In particular, it must be clear if one wishes to assure that storage of −0 by the FP11-E is always accompanied by an interrupt.
2. Through the rest of this chapter, it is assumed that the FID bit is clear in all discussions involving overflow, underflow, occurrence of −0, and integer conversion errors.

13	Not Used	
12	Not used	
11	Interrupt on Undefined Variable (FIUV)	An interrupt occurs if FIUV is set and a −0 is obtained from memory as an operand of ADD, SUB, MUL, DIV, CMP, MOD, NEG, ABS, TST or any LOAD instruction. The interrupt occurs before execution on the FP11-E except on NEG and ABS for which

BIT	NAME	DESCRIPTION
		it occurs after execution. When FIUV is reset, —0 can be loaded and used in any FPP operation. Note that the interrupt is not activated by the presence of —0 in an AC operand of an arithmetic instruction: in particular, trap on —0 never occurs in Mode 0. The FP11-E will not store a result of —0 without the simultaneous occurrence of an interrupt (See Section 10.4).
10	Interrupt on Underflow (FIU)	When the FIU bit is set, Floating Underflow will cause an interrupt. The fractional part of the result of the operation causing the interrupt will be correct. The biased exponent will be too large by 400 (octal), except for the special case of 0, which is correct. An exception is discussed in the detailed description of the LDEXP instruction. If the FIU bit is reset and if underflow occurs, no interrupt occurs and the result is set to exact 0.
9	Interrupt on Overflow (FIV)	When the FIV bit is set, Floating Overflow will cause an interrupt. The fractional part of the result of the operation causing the overflow will be correct. The biased exponent will be too small by 400 (octal). If the FIV bit is reset, and overflow occurs, there is no interrupt. The FP11-E returns exact 0. Special cases of overflow are discussed in the detailed descriptions of the MOD and LDEXP instructions.
8	Interrupt on Integer Conversion Error (FIC)	When the FIC bit is set, and a conversion to integer instruction fails, an interrupt will occur. If

BIT	NAME	DESCRIPTION
		the interrupt occurs, the destination is set to 0, and all other registers are left untouched. If the FIC bit is reset, the result of the operation will be the same as detailed above, but no interrupt will occurr. The conversion instruction fails if it generates an integer with more bits than can fit in the short or long integer word specified by the FL bit (see 6 below).
7	Floating Double Precision Mode (FD)	Determines the precision that is used for floating point calculations. When set, double precision is assumed; when reset, single precision is used.
6	Floating Long Integer Mode (FL)	Active in conversion between integer and floating point format. When set, the integer format assumed is double precision two's complement (i.e. 32 bits). When reset, the integer format is assumed to be single precision two's complement (i.e. 16 bits).
5	Floating Chop Mode (FT)	When bit FT is set, the result of any arithmetic operation is chopped (or truncated). When reset, the result is rounded. See Section 10.8 for a discussion of the chopping and rounding operations.
4	Floating Maintenance Mode (FMM)	This code is a maintenance feature. Refer to the Maintenance Manual for the details of its operation. The FMM bit can be set only in Kernel Mode.
3	Floating Negative (FN)	FN is set if the result of the last operation was negative, otherwise it is reset.
2	Floating Zero (FZ)	FZ is set if the result of the last operation was zero; otherwise it is reset.
1	Floating Overflow (FV)	FV is set if the last operation resulted in an exponent overflow; otherwise it is reset.

BIT	NAME	DESCRIPTION
0	Floating Carry (FC)	FC is set if the last operation resulted in a carry of the most significant bit. This can only occur in floating or double to integer conversions.

10.6 FLOATING EXCEPTION CODE AND ADDRESS REGISTER

One interrupt vector is assigned to take care of all floating point exceptions (location 244). The seven possible errors are coded in the four bit FEC (Floating Exception Code) register as follows:

2 Floating OP code error
4 Floating divide by zero
6 Floating (or double) to integer conversion error
8 Floating overflow
10 Floating underflow
12 Floating undefined variable
14 Maintenance trap

The address of the instruction producing the exception is stored in the FEA (Floating Exception Address) register.

The FEC and FEA registers are updated only when one of the following occurs:

1. divide by zero
2. illegal OP code*
3. any of the other five exceptions with the corresponding interrupt is enabled.

NOTE

1. If one of the last five exceptions occurs with the corresponding interrupt disabled, the FEC and FEA are not updated.
2. Inhibition of interrupts by the FID bit does not inhibit updating of the FEC and FEA, if an exception occurs.
3. The FEC and FEA do not get updated if no exception occurs. This means that the STST (store status) instruction will return current information only if the most recent floating point instruction produced an exception.
4. Unlike the FPS register, no instructions are provided for storage into the FEC and FEA registers.

10.7 FLOATING POINT PROCESSOR INSTRUCTION ADDRESSING

Floating Point Processor instructions use the same type of addressing as the Central Processor instructions. A source or destination operand is specified by designating one of eight addressing modes and one of eight central processor general registers to be used in the specified mode. The modes of addressing are the same as those of the central processor except for mode 0. In mode 0 the operand is located in the designated Floating Point Processor Accumulator, rather than in a Central processor general register. The modes of addressing:

0 = Direct Accumulator

1 = Deferred

2 = Auto-increment

3 = Auto-increment deferred

4 = Auto-decrement

5 = Auto-decrement deferred

6 = Indexed

7 = Indexed deferred

Autoincrement and autodecrement operate on increments and decrements of 4 for F Format and 10_8 for D Format.

In mode 0, the user can make use of all six FPP accumulators (AC0—AC5) as his source or destination. In all other modes, which involve transfer of data from memory or the general register, the user is restricted to the first four FPP accumulators (AC0—AC3).

In immediate addressing (Mode 2, R7) only 16 bits are loaded or stored.

10.8 ACCURACY

General comments on the accuracy of the FPP are presented here. The descriptions of the individual instructions include the accuracy at which they operate. An instruction or operation is regarded as "exact" if the result is identical to an infinite precision calculation involving the same operands. The a priori accuracy of the operands is thus ignored. All arithmetic instructions treat an operand whose biased exponent is 0 as an exact 0 (unless FIUV is enabled and the operand is −0, in which case an interrupt occurs). For all arithmetic operations, except DIV, a zero operand implies that the instruction is exact. The same statement holds for DIV if the zero operand is the dividend. But if it is the divisor, division is undefined and an interrupt occurs.

For non-vanishing floating point operands, the fractional part is binary normalized. It contains 24 bits or 56 bits for Floating Mode and Double Mode, respectively. The internal hardware registers contain 60 bits for processing the fractional parts of the operands, of which the high order bit is reserved for arithmetic overflow. Therefore there are, internally, 35 guard bits for Floating Mode and 3 guard bits for Double Mode arithmetic operations. For ADD, SUB, MUL, and DIV, two guard bits are necessary and sufficient to guarantee return of a chopped or rounded result identical to the corresponding infinite precision operation chopped or rounded to the specified word length. Thus, with two guard bits, a chopped result has an error bound of one least significant bit (LSB); a rounded result has an error bound of 1/2 LSB. (For a radix other than 2, replace "bit" with "digit" in the two preceding sentences to get the corresponding statements on accuracy.) These error bounds are realized by the FP11-E for most instructions. For the addition of operands of opposite sign or for the subtraction of operands of the same sign in rounded double precision, the error bound is 3/4 LSB, which is slightly larger than the 1/2 LSB error bound for all other rounded operations.

In the rest of this chapter an arithmetic result is called exact if no non-vanishing bits would be lost by chopping. The first bit lost in chopping

is referred to as the "rounding"bit. The value of a rounded result is related to the chopped result as follows:

1. if the rounding bit is one, the rounded result is the chopped result incremented by an LSB (least significant bit).
2. if the rounding bit is zero, the rounded and chopped results are identical.

It follows that

1. If the result is exact
   rounded value = chopped value = exact value
2. If the result is not exact, its magnitude
   (a) is always decreased by chopping
   (b) is decreased by rounding if the rounding bit is zero
   (c) is increased by rounding if the rounding bit is one.

Occurrence of floating point overflow and underflow is an error condition: the result of the calculation cannot be correctly stored because the exponent is too big to fit into the 8 bits reserved for it. However, the internal hardware has produced the correct answer. For the case of underflow replacement of the correct answer by zero is a reasonable resolution of the problem for many applications. This is done on the FP11-E if the underflow interrupt is disabled. The error incurred by this action is an absolute rather than a relative error; it is bounded (in absolute value) by 2**(—128). There is no such simple resolution for the case of overflow. The action taken, if the overflow interrupt is disabled, is described under FIV (bit 9) of Section 10.5.

The FIV and FIU bits (of the floating point status word) provide the user with an opportunity to implement his own fix up of an overflow or underflow condition. If such a condition occurs and the corresponding interrupt is enabled, the hardware stores the fractional part and the low eight bits of the biased exponent. The interrupt will take place and the user can identify the cause by examination of the FV (floating overflow) bit or the FEC (floating exception) register. The reader can readily verify that (for the standard arithmetic operations ADD, SUB, MUL, and DIV) the biased exponent returned by the hardware bears the following relation to the correct exponent generated by the hardware:

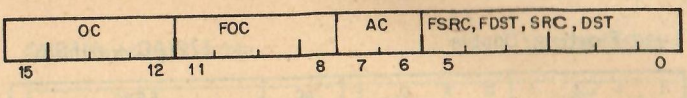
1. on overflow: it is too small by 400 octal
2. on underflow: if the biased exponent is 0 it is correct. If it is not 0, it is too large by 400 octal.

Thus, with the interrupt enabled, enough information is available to determine the correct answer. The user may, for example, rescale his variables (via STEXP and LDEXP) to continue his calculation. Note that the accuracy of the fractional part is unaffected by the occurrence of underflow or overflow.

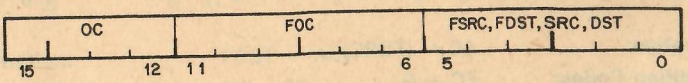
10.9 FLOATING POINT INSTRUCTIONS

Each instruction that references a floating point number can operate on either floating or double precision numbers depending on the state of the FD mode bit. Similarly, there is a mode bit FL that determines whether a 32-bit integer (FL = 1) or a 16-bit integer (FL = 0)is used in conversion between integer and floating point representation. FSRC and FDST use floating point addressing modes; SRC and DST use CPU addressing Modes.

Floating Point Instruction Format
Double Operand Adressing



Single Operand Addressing

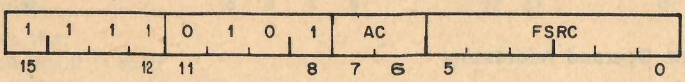


OC = Op Code = 17
FOC = Floating Op Code
AC = Accumulator
FSRC, FDST use FPP Address Modes
SRC, DST use CPU Address Modes

General Definitions:
XL = largest fraction that can be represented:
1−2**(−24), FD = 0; single precision
1−2**(−56), FD = 1); double precision
XLL = smallest number that is not identically zero = 2**(−128) −2**(−127))*(1/2)
XUL = largest number that can be represented = 2**(127)*XL
JL = largest integer that can be represented:
2**(15)−1 if FL = 0 2**(31)−1 if FL = 1
ABS (address) = absolute value of (address)
EXP (address) = biased exponent of (address)
.LT. = "less than"
.LE. = "less than or equal"
.GT. = "greater than"
.GE. = "greater than or equal"
LSB = least significant bit

LDF
LDD

Load Floating/Double 172(AC + 4)FSRC



Operation: AC ← (FSRC)

Condition Codes:
FC ← 0
FV ← 0
FZ ← 1 if (AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

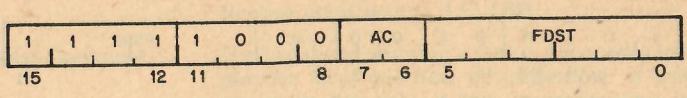
Description: Load Single or Double Precision Number into Accumulator.

Interrupts: If FIUV is enabled, trap on −0 occurs before AC is loaded. However, the condition codes will reflect a fetch of minus 0 regardless of the FIUV bit. Neither overflow nor underflow can occur.

Accuracy: These instructions are exact.

Special Comment: These instructions permit use of −0 in a subsequent floating point instruction if FIUV is not enabled and (FSRC) = −0.

Store Floating/Double 174ACFDST



Operation: FDST ← (AC)

Condition Codes:
FC ← FC
FV ← FV
FZ ← FZ
FN ← FN

Description: Store Single or Double Precision Number from Accumulator.

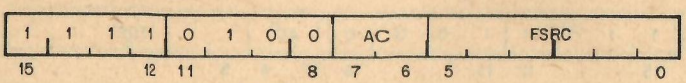
Interrupts: These instructions do not interrupt if FIUV enabled, because the —0, if present, is in AC, not in memory. Neither overflow nor underflow can occur.

Accuracy: These instructions are exact.

Special Comment: These instructions permit storage of a —0 in memory from AC. There are two conditions in which minus 0 can be stored in AC of the FP11-C or FP11-E. One occurs when underflow or overflow is present and the corresponding interrupt is enabled. A second occurs when an LDI or LDF instruction is executed and the FIUV bit is disabled.

ADDF
ADDD

Add Floating/Double 172ACFSRC



Operation:

Let SUM = (AC) + (FSRC):

If underflow occurs and FIU is not enabled, AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0.

For all other cases, AC ← SUM.

Condition Codes:

FC ← 0.
FV ← 1 If overflow occurs, else FV ← 0.
FZ ← 1 If (AC) = 0, else FZ ← 0.
FN ← 1 If (AC) < 0, else FN ← 0.

Description:

Add the contents of FSRC to the contents of AC. The addition is carried out in single or double precision and is rounded or chopped in accordance with the values of the FD and FT bits in the FPS register. The result is stored in AC except for overflow with interrupt disabled and underflow with interrupt disabled.

For these exceptional cases an exact 0 is stored in AC.

Interrupts:

If FIUV is enabled, trap on −0 in FSRC occurs before execution.

If overflow or underflow occurs and if the corresponding interrupt is enabled, the trap occurs with the faulty result in AC. The fractional parts are correctly stored. The exponent part is too large by 400 octal for underflow, except for the special case of 0, which is correct.

Accuracy:

Errors due to overflow and underflow are described above. If neither occurs, then: For oppositely signed operands with exponent differences of 0 or 1, the answer returned is exact if a loss of significance of one or more bits occurs. Note that these are the only cases for which loss of significance of more than one bit can occur. For all other cases the result is inexact with error bounds of

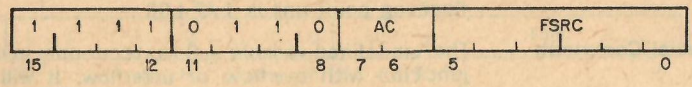
1 LSB in chopping mode with either single or double precision.

3/4 LSB in rounding mode with double precision. For an ADD instruction specifying double precision with rounding, the accuracy of the integral floating point unit is 1/2 LSB.

Special Comment: The undefined variable —0 can occur only in conjunction with overflow or underflow. It will be stored in AC only if the corresponding interrupt is enabled.

SUBF
SUBD

Subtract Floating/Double 173ACFSRC



Operation:

Let DIFF = (AC) − (FSRC):

If underflow occurs and FIU is not enabled, AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0.

For all other cases, AC ← DIFF.

Condition Codes:

FC ← 0.
FV ← 1 If overflow occurs, else FV ← 0.
FZ ← 1 If (AC) = 0, else FZ ← 0.
FN ← 1 If (AC) < 0, else FN ← 0.

Description:

Subtract the contents of FSRC from the contents of AC. The subtraction is carried out in single or double precision and is rounded or chopped in accordance with the values of the FD and FT bits in the FPS register. The result is stored in AC except for: overflow with interrupt disabled and underflow with interrupt disabled.

For these exceptional cases, an exact 0 is stored in AC.

Interrupts:

If FIUV is enabled, trap on −0 in FSRC occurs before execution.

If overflow or underflow occurs and if the corresponding interrupt is enabled, the trap occurs with the faulty results in AC. The fractional parts are correctly stored. The exponent part is too small by 400 octal for overflow. It is too large by 400 octal for underflow, except for the special case of 0, which is correct.

Accuracy:

Errors due to overflow and underflow are described above. If neither occurs, then: For like-signed operands with exponent difference of 0 or 1, the answer returned is exact if a loss of significance of more than one bit can occur. Note that these are the only cases for which loss of significance of more than one bit can occur. For all other cases the result is inexact with error bounds of 1 LSB in chopping mode with either single or double precision.

1/2 LSB in rounding mode with single precision.

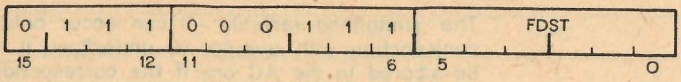
3/4 LSB in rounding mode with double precision. For the SUB instruction specifying double precision with rounding, the accuracy of the integral floating point unit is 1/2 LSB.

Special Comment:

The undefined variable —0 can occur only in conjunction with overflow or underflow. It will be stored in the AC only if the corresponding interrupt is enabled.

NEGF
NEGD

Negate Floating/Double 1707FDST



Operation: FDST ← −(FDST) if EXP(FDST) ≠ 0, else FDST ← exact 0.

Condition Codes:
FC ← 0.
FV ← 0.
FZ ← 1 If EXP(FDST) = 0, else FZ ← 0.
FN ← 1 If (FDST) < 0, else FN ← 0.

Description: Negate single or double Precision number, store result in same location. (FDST)

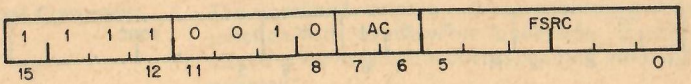
Interrupts: If FIUV is enabled
Trap on −0 occurs after execution.

Neither overflow nor underflow can occur.

Accuracy: These instructions are exact.

Special Comment: If a minus 0 is present in memory and the FIUV bit is enabled, then the FP11-E and the integral floating point unit store exact 0 in memory (zero exponent, zero fraction, and positive sign). The condition code reflects an exact 0 (FZ ← 1).

Multiply Floating/Double 171ACFSRC



Operation:

Let PROD = (AC)*(FSRC)

If underflow occurs and FIU is not enabled, AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0.

For all other cases AC ←PROD

Condition Codes:

FC ← 0.
FV ← 1 if overflow occurs, else FV ← 0.
FZ ← 1 if (AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description:

If the biased exponent of either operand is zero, (AC) ← exact 0. For all other cases PROD is generated to 48 bits for Floating Mode and 59 bits for Double Mode. The product is rounded or chopped for FT = 0 and 1, respectively, and is stored in AC except for

Overflow with interrupt disabled.

Underflow with interrupt disabled.

For these exceptional cases, an exact 0 is stored in accumulator.

Interrupts:

If FIUV is enabled, trap on −0 occurs before execution.

If overflow or underflow occurs and if the corresponding interrupt is enabled, the trap occurs with the faulty results in AC. The fractional parts are correctly stored. The exponent part is too small by 400 octal for overflow. It is too large by 400 octal for underflow, except for the special case of 0, which is correct.

Accuracy:

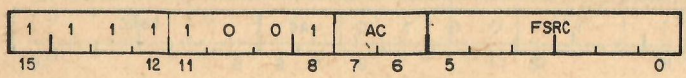
Errors due to overflow and underflow are described above. If neither occurs, the error incurred is bounded by 1 LSB in chopping mode and 1/2 LSB in rounding mode.

Special Comment:

The undefined variable −0 can occur only in conjunction with overflow or underflow. It will be stored in AC only if corresponding interrupt is enabled.

DIVF
DIVD

Divide Floating/Double 174(AC + 4)FSRC



Operation:

If EXP(FSRC) = 0, AC ← (AC): instruction is aborted.

If EXP(AC) = 0, AC ← exact 0.

For all other cases, let QUOT = (AC)/(FSRC):

If underflow occurs and FIU is not enabled AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0.

For all remaining cases AC ← QUOT.

Condition Codes:

FC ← 0.
FV ← 1 if overflow occurs, else FV ← 0.
FZ ← 1 if EXP(AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description:

If either operand has a biased exponent of 0, it is treated as an exact 0. For FSRC this would imply division by zero; in this case the instruction is aborted, the FEC register is set to 4 and an interrupt occurs. Otherwise the quotient is developed to single or double precision with enough guard bits for correct rounding. The quotient is rounded or chopped in accordance with the values of the FD and FT bits in the FPS register. The result is stored in AC except for:

Overflow with interrupt disabled.

Underflow with interrupt disabled.

For these exceptional cases an exact 0 is stored in accumulator.

Interrupts:

If FIUV is enabled, trap on −0 in FSRC occurs before execution.

If EXP(FSRC) = 0 interrupt traps on attempt to divide by 0.

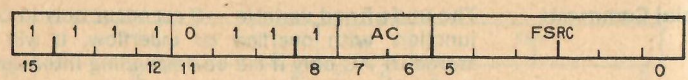
If overflow or underflow occurs and if the corresponding interrupt is enabled, the trap occurs with the faulty results in AC. The fractional parts are correctly stored. The exponent part is too small by 400 octal for overflow. It is too large by 400 octal for underflow, except for the special case of 0, which is correct.

Accuracy: Errors due to overflow, underflow and division by 0 are described above. If none of these occurs, the error in the quotient will be bounded by 1 LSB in chopping mode and by 1/2 LSB in rounding mode.

Special Comment: The undefined variable —0 can occur only in conjunction with overflow or underflow. It will be stored in AC only if the corresponding interrupt is enabled.

CMPF
CMPD

Compare Floating/Double 173 (AC + 4) FSRC



Operation: (FSRC) — (AC)

Condition Codes:
FC ← 0.
FV ← 0.
FZ ← 1 If (FSRC) — (AC) = 0, else FZ ← 0.
FN ← 1 If (FSRC) — (AC) < 0, else FN ← 0.

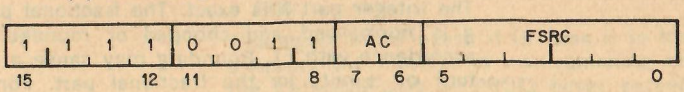
Description: Compare the contents of FSRC with the accumulator. Set the appropriate floating point condition codes. FSRC and the accumulator are left unchanged.

Interrupts: If FIUV is enabled, trap on —0 occurs before execution.

Accuracy: These instructions are exact.

Special Comment: An operand which has a biased exponent of zero is treated as if it were exact zero. In the case where both operands are zero, the floating point processor will store exact 0 in the AC.

Multiply and Integerize Floating/Double 171(AC + 4)FSRC



Description and Operation

This instruction generates the product of its two floating point operands, separates the product into integer and fractional parts and then stores one or both parts as floating point numbers.

Let PROD = (AC)*(FSRC) so that in:

Floating point: ABS(PROD) = (2**K) *f

where 1/2.LE.f.LT.1 and
EXP(PROD) = (200 + K) octal

Fixed Point binary: PROD = N + g, with

N = INT(PROD) = the integer part of PROD

and

g = PROD − INT(PROD) = the fractional part of PROD with O.LE.g.LT.1

Both N and g have the same sign as PROD. They are returned as follows:

If AC is an even-numbered accumulator (0 or 2), N is stored in AC + 1 (1 or 3), and g is stored in AC.

If AC is an odd-numbered accumulator, N is not stored, and g is stored in AC.

The two statements above can be combined as follows: N is returned to ACv1 and g is returned to AC, where v means .OR.

Five special cases occur, as indicated in the following formal description with L = 24 for Floating Mode and L = 56 for Double Mode:

1. If PROD overflows and FIV enabled:

   ACv1 ← N, chopped to L bits, AC ← exact 0

Note that EXP(N) is too small by 400 (octal), and that ←0 can get stored in ACv1.

If FIV is not enabled: ACv1 ← exact 0, AC ← exact 0, and −0 will never be stored.

2. If 2**L.LE.ABS(PROD) and no overflow

   ACv1 ← N, chopped to L bits, AC ← exact 0

The sign and EXP of N are correct, but low order bit information, such as parity, is lost.

3. If 1.LE.ABS(PROD).LT.2**L

   ACv1 ← N, AC ← g

The integer part N is exact. The fractional part g is normalized, and chopped or rounded in accordance with FT. Rounding may cause a return of ±unity for the fractional part. For L = 24, the error in g is bounded by 1 LSB in chopping mode and by 1/2 LSB in rounding mode. For L = 56, the error in g increases from the above limits as ABS(N) increases above 3 because only 59 bits of PROD are generated:

if 2**p.LE.ABS(N).LT.2**(p + 1), with p > 2, the low order p − 2 bits of g may be in error.

4. If ABS (PROD). LT.1 and no underflow:

   ACv1 ← exact 0 AC ← g

There is no error in the integer part. The error in the fractional part is bounded by 1 LSB in chopping mode and 1/2 LSB in rounding mode. Rounding may cause a return of ±unity for the fractional part.

5. If PROD underflows and FIU enabled:

   ACv1 ← exact 0 AC ← g

Errors are as in case 4, except that EXP(AC) will be too large by 400 octal (except if EXP = 0, it is correct). Interrupt will occur and −0 can be stored in AC.

IF FIU is not enabled, ACv1 ← exact 0 and AC ← exact 0. For this case the error in the fractional part is less than 2**(−128).

Condition Codes:

FC ← 0.
FV ← 1 if PROD overflows, else FV ← 0.
FZ ← 1 if (AC) = 0, else FZ ← 0.
FN ← if (AC) < 0, else FN ← 0.

Interrupts:

If FIUV is enabled, trap on −0 in FSRC will occur before execution.

Overflow and Underflow are discussed above.

Accuracy:

Discussed above.

Applications:

1. Binary to decimal conversion of a proper fraction: the following algorithm, using MOD, will generate decimal digits D(1), D(2) . . . from left to right:

Initialize: I ← 0
X ← number to be converted;
ABS(X) < 1

```
While X ≠ 0 do
Begin PROD ← X*10;
  I ← I + 1;
  D(I) ← INT(PROD);
  X ← PROD — INT(PROD);
  END;
```

This algorithm is exact; it is case 3 in the description: the number of non-vanishing bits in the fractional part of PROD never exceeds L, and hence neither chopping nor rounding can introduce error.

2. To reduce the argument of a trigonometric function.

ARG*2/PI = N + g. The low two bits of N identify the quadrant, and g is the argument reduced to the first quadrant. The accuracy of N + g is limited to L bits because of the factor 2/PI. The accuracy of the reduced argument thus depends on the size of N.

3. To evaluate the exponential function e**x, obtain

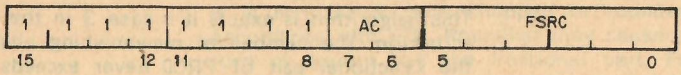
   x*(log e base 2) = N + g.
   Then e**x = (2**N)*(e**(g*ln 2))

The reduced argument is g*ln2 < 1 and the factor 2**N is an exact power of 2, which may be scaled in at the end via STEXP, ADD N to EXP and LDEXP. The accuracy of N + g is limited to L bits because of the factor (log e base 2). The accuracy of the reduced argument thus depends on the size of N.

LDCDF
LDCFD

Load and convert from Double to Floating or from Floating to Double

177(AC + 4)FSRC



Operation:

If EXP(FSRC) = 0, AC ← exact 0.

If FD = 1, FT = 0, FIV = 0 and rounding causes overflow, AC ← exact 0.

In all other cases AC ← C_{xy} (FSRC), where C_{xy} specifies conversion from floating mode x to floating mode y;

x = D, y = F if FD = 0 (single)
x = F, y = D if FD = 1. (double)

Condition Codes:

FC ← 0.
FV ← 1 if conversion produces overflow, else FV ← 0.
FZ ← 1 if (AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description:

If the current mode is Floating Mode (FD = 0) the source is assumed to be a double-precision number and is converted to single precision. If the Floating Chop bit (FT) is set, the number is chopped, otherwise the number is rounded.

If the current mode is Double Mode (FD = 1), the source is assumed to be a single-precision number, and is loaded left justified in the AC. The lower half of the AC is cleared.

Interrupts:

If FIUV is enabled, trap on −0 occurs before execution. However, the condition codes will reflect a fetch minus 0 regardless of the FIUV bit.

Overflow cannot occur for LDCFD.

A trap occurs if FIV is enabled, and if rounding with LDCDF causes overflow; AC ← overflowed result of conversion. This result must be +0 or −0.

Underflow cannot occur.

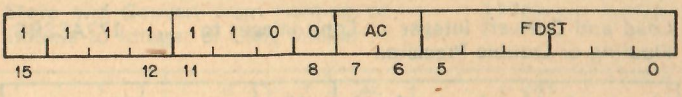
Accuracy:

LDCFD is an exact instruction. Except for overflow, described above, LDCDF incurs an error bounded by one LSB in chopping mode, and by 1/2 LSB in rounding mode.

STCFD
STCDF

Store and convert from Floating to Double or from Double to Floating

176ACFDST



Operation:

If EXP(AC) = 0, FDST ← exact 0

If FD = 1, FT = 0, FIV = 0 and rounding causes overflow, FDST ← exact 0.

In all other cases, FDST ← C_{xy}(AC), where C_{xy} specifies conversion from floating mode x to floating mode y;

x = F and y = D if FD = 0,
x = D and y = F if FD = 1.

Condition Codes:

FC ← 0.
FV ← 1 If conversion produces overflow else FV ← 0.
FZ ← 1 If (AC) = 0, else FZ ← 0.
FN ← 1 If (AC) < 0, else FN ← 0.

Description:

If the current mode is single precision, the Accumulator is stored left justified in FDST and the lower half is cleared. If the current mode is double precision, the contents of the accumulator are converted to single precision, chopped or rounded depending on the state of FT, and stored in FDST.

Interrupts:

Trap on −0 will not occur even if FIUV is enabled because FSRC is an accumulator.

Underflow cannot occur.

Overflow cannot occur for STCFD.

A trap occurs if FIV is enabled, and if rounding with STCDF causes overflow; FDST ← overflowed result of conversion. This result must be +0 or −0.

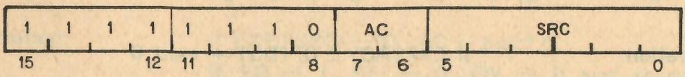
Accuracy:

STCFD is an exact instruction. Except for overflow, described above, STCDF incurs an error bounded by 1 LSB in chopping mode and 1/2 LSB in rounding mode.

LDCIF
LDCID
LDCLF
LDCLD

Load and Convert Integer or Long Integer to Floating or Double Precision

177ACSRC



Operation:

AC ← C_{jx} (SRC), where
C_{jx} specifies conversion from integer mode j to floating mode x;

j = I if FL = 0, j = L if FL = 1,
x = F if FD = 0, x = D if FD = 1.

Condition Codes:

FC ← 0.
FV ← 0.
FZ ← 1 If (AC) = 0, else FZ ← 0.
FN ← 1 If (AC) < 0, else FN ← 0.

Description:

Conversion is performed on the contents of SRC from a 2's complement integer with precision j to a floating point number of precision x. Note that j and x are determined by the state of the mode bits FL and FD: J = I or L, and X = F or D.

If a 32-bit Integer is specified (L mode) and (SRC) has an addressing mode of 0, or immediate addressing mode is specified, the 16 bits of the source register are left justified and the remaining 16 bits loaded with zeroes before conversion.

In the case of LDCLF the fractional part of the floating point representation is chopped or rounded to 24 bits for FT = 1 and 0 respectively.

Interrupts:

None; SRC is not floating point, so trap on −0 cannot occur.

Overflow and underflow cannot occur.

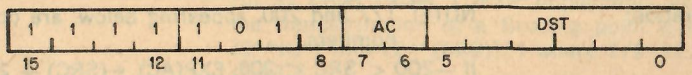
Accuracy:

LDCIF, LDCID, LDCLD are exact instructions. The error incurred by LDCLF is bounded by one LSB in chopping mode, and by 1/2 LSB in rounding mode.

STCFI
STCFL
STCDI
STCDL

Store and Convert from Floating or Double to Integer or Long Integer

175(AC + 4)DST



Operation:

DST ← C_{xj} (AC) if $-JL-1 < C_{xj}$ (AC) $< JL+1$, else DST ← 0, where C_{xj} specifies conversion from floating mode x to integer mode j;

j = I if FL = 0, j = L if FL = 1,
x = F if FD = 0, x = D if FD = 1.

JL is the largest integer:

2**15 − 1 for FL = 0
2**31 − 1 for FL = 1

Condition Codes:

C ← FC ← 0 if $-JL-1 < C_{xj}$ (AC) $< JL+1$, else FC ← 1.
V ← FV ← 0.
Z ← FZ ← 1 if (DST) = 0, else FZ ← 0.
N ← FN ← 1 if (DST) < 0, else FN ← 0.

Description:

Conversion is performed from a floating point representation of the data in the accumulator to an integer representation.

If the conversion is to a 32-bit word (L mode) and an address mode of 0, or immediate adressing mode, is specified, only the most significant 16 bits are stored in the destination register.

If the operation is out of the integer range selected by FL, FC is set to 1 and the contents of the DST are set to 0.

Numbers to be converted are always chopped (rather than rounded) before conversion. This is true even when the Chop Mode bit, FT is cleared in the Floating Point Status Register.

Interrupts:

These instructions do not interrupt if FIUV is enabled, because the −0, if present, is in AC, not in memory.

If FIC enabled, trap on conversion failure will occur.

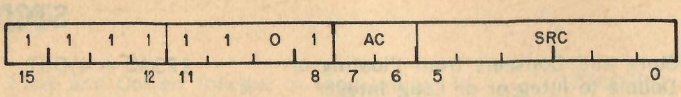
Accuracy:

These instructions store the integer part of the floating point operand, which may not be the integer most closely approximating the operand. They are exact if the integer part is within the range implied by FL.

LDEXP

Load Exponent

1776(AC + 4)SRC



Operation:

NOTE: 177 and 200, appearing below, are octal numbers.

If −200 < SRC < 200, EXP(AC) ←(SRC) + 200 and the rest of AC is unchanged on FP11C and FP11B.

If (SRC) > 177 and FIV is enabled,
EXP(AC) ← (SRC) < 6:0 >
EXP(AC) ← (SRC) < 7:0 > on FP11B.

If (SRC) > 177 and FIV is disabled,
AC ← exact 0
EXP(AC) ← (SRC + 200) < 7:0 > on FP11B.

If (SRC) < −177 and FIU is disabled,
AC ← exact 0.

If (SRC) < −177 and FIU is enabled,
EXP(AC) ← (SRC) < 6:0 >
EXP(AC) ← (SRC)+ 200) < 7:0 >

Condition Codes:

FC ← 0.
FV ← 1 if (SRC) > 177, else FV ← 0.
FZ ← 1 if EXP(AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description:

Change AC so that its unbiased exponent = (SRC). That is, convert (SRC) from 2's complement to excess 200 notation, and insert in the EXP field of AC. This is a meaningful operation only if ABS(SRC).LE.177.

If SRC > 177, result is treated as overflow. If SRC < −177, result is treated as underflow. Note that the FP11C and FP11B do not treat these abnormal conditions in exactly the same way.

Interrupts:

No trap on −0 in AC occurs, even if FIUV enabled.

If SRC > 177 and FIV enabled, trap on overflow will occur.

If SRC < −177 and FIU enabled, trap on underflow will occur.

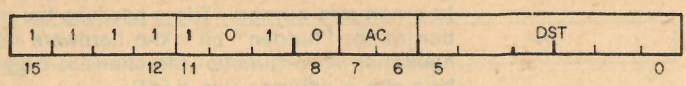
Accuracy:

Errors due to overflow and underflow are described above. If EXP(AC) = 0 and SRC ≠ −200, (AC) changes from a floating point number treated as 0 by all floating arithmetic operations to a non-zero number. This is because the insertion of the "hidden" bit in the hardware implementation of arithmetic instructions is triggered by a non-vanishing value of EXP.

For all other cases, LDEXP implements exactly the transformation of a floating point number (2**K)*f into (2**(SRC))*f where 1/2 .LE.ABS (f).LT.1.

STEXP

Store Exponent 175ACDST



Operation: DST ← EXP(AC)—200 octal

Condition Codes:
C ← FC ← 0.
V ← FV ← 0.
Z ← FZ ← 1 if (DST) = 0, else FZ ← 0.
N ← FN ← 1 if (DST) < 0, else FN ← 0.

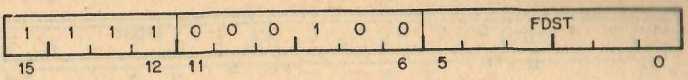
Description: Convert accumulator's exponent from excess 200 octal notation to 2's complement, and store result in DST.

Interrupts: This instruction will not trap on —0.

Overflow and underflow cannot occur.

Accuracy: This instruction is always exact.

Clear Floating/Double 1704FDST



Operation: FDST ← exact 0.

Condition Codes:
FC ← 0.
FV ← 0.
FZ ← 1
FN ← 0.

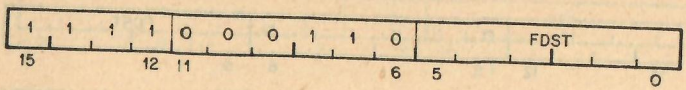
Description: Set FDST to 0. Set FZ condition code and clear other condition code bits.

Interrupts: No interrupts will occur. Neither overflow nor underflow can occur.

Accuracy: These instructions are exact.

ABSF
ABSD

Make Absolute Floating/Double 1706FDST



Operation: If (FDST) < 0, FDST ← — (FDST).
If EXP(FDST) = 0, FDST ← exact 0.
For all other cases, FDST ← (FDST).

Condition Codes: FC ← 0.
FV ← 0.
FZ ← 1 if EXP(FDST) = 0, else FZ ← 0.
FN ← 0

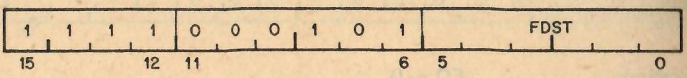
Description: Set the contents of FDST to its absolute value.

Interrupts: If FIUV is set. Trap on —0 occurs after execution.

Accuracy: These instructions are exact.

Special Comment: If a minus 0 is present in memory and the FIUV bit is enabled, then the FP11-E and integral floating point unit store exact 0 in memory (zero exponent, zero fraction, and positive sign). The condition code reflects an exact 0 (FZ ← 1).

Test Floating/Double 1705FDST



Operation: FDST ← (FDST)

Condition Codes:
FC ← 0.
FV ← 0.
FZ ← 1 if EXP(FDST) = 0, else FZ ← 0.
FN ← 1 if (FDST) < 0, else FN ← 0.

Description: Set the Floating Point Processor's Condition Codes according to the contents of FDST.

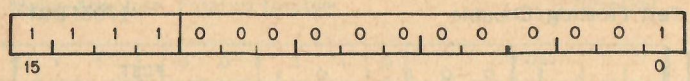
Interrupts: If FIUV is set, trap on —0 occurs after execution

Overflow and underflow cannot occur.

Accuracy: These instructions are exact.

SETF

Set Floating Mode 170001

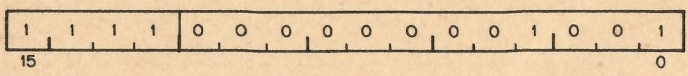


Operation: FD ← 0

Description: Set the FPP in Single Precision Mode.

SETD

Set Floating Double Mode 170011

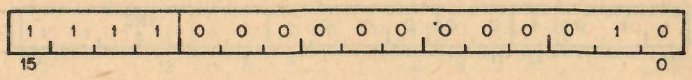


Operation: FD ← 1

Description: Set the FPP in Double Precision Mode.

SETI

Set Integer Mode 170002

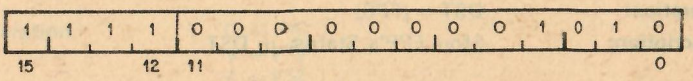


Operation: FL ← 0

Description: Set the FPP for Integer Data.

SETL

Set Long Integer Mode 170012



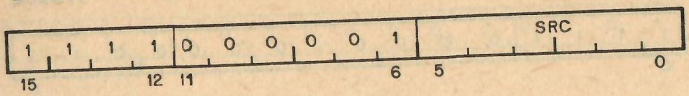
Operation: FL ← 1

Description: Set the FPP for Long Integer Data.

LDFPS

Load FPPs Program Status

1701SRC



Operation: FPS ← (SRC)

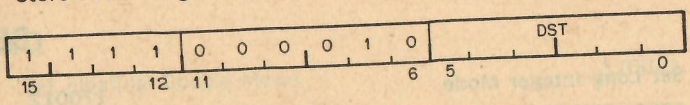
Description: Load FPP's Status from SRC.

Special Comment: The user is cautioned not to use bits 13 and 12 for his own purposes, since these bits are not recoverable by the STFPS instruction.

STFPS

Store FPPs Program Status

1702DST



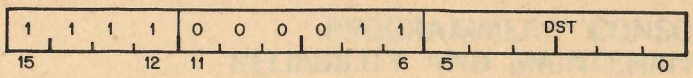
Operation: DST ← (FPS)

Description: Store FPP's Status in DST.

Special Comment: Bits 13 and 12 are loaded with 0. All other bits are the corresponding bits in the FPS.

STST

Store FPPs Status 1703DST



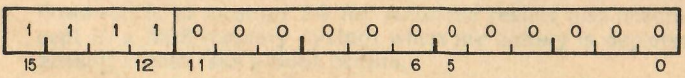
Operation: DST ← (FEC)
DST + 2 ←(FEA)

Description: Store the FEC and then the FPP's Exception Address Pointer in DST and DST + 2.

NOTES:
1. If destination mode specifies a general register or immediate addressing, only the FEC is saved.
2. The information in these registers is current only if the most recently executed floating point instruction (refer to Section 10.6) caused a floating point exception.

CFCC

Copy Floating Condition Codes 170000



Operation: C ← FC
V ← FV
Z ← FZ
N ← FN

Description: Copy FPP Condition Codes into the CPU's Condition Codes.

CHAPTER 11

PROGRAMMERS' CONSOLE RELIABILITY AND MAINTENANCE

11.1 INTRODUCTION—CONSOLE

The KY11-P Programmers' Console is utilized for both computer operation and maintenance. Within the maintenance function, the console supplements other PDP-11/60 features such as single clock, microbreak, processor error log, error status registers and device-specific macrodiagnostics. Microdiagnostics are also available with the PDP-11/60 via the Diagnostic Control Store (KU116-BB).

11.2 KY11-P PROGRAMMERS' CONSOLE

The PDP-11/60 console (Figure 11-1) allows direct control of the computer system. It contains a power switch which is used as the master switch for the system. The console is used for starting, stopping, resetting and debugging programs. Lights and switches provide the facilities for monitoring operation, system control and maintenance. Debugging and detailed tracing of operations can be accomplished by having the computer execute single instructions. Contents of all memory locations and internal registers can be examined and data entered manually from the console switches.

11.2.1 Power-up

Power is turned on by turning the Rotary Switch to POWER. What occurs after power-up depends on the position of the BOOT/RUN/HALT slide switch prior to the power-up. The slide switch allows three modes of power-up: BOOT, RUN, and HALT.

BOOT: This position allows the system to boot directly from the Bootstrap Loader (M9301-YX). The boot is accomplished by selecting the device to be bootstrapped by microswitches on the M9301-YX, placing the slide switch in BOOT position and turning the rotary switch to POWER.

RUN: This position allows automatic restart on power-fail recovery. Power-up is to location 24 for automatic restart and occurs except in a MOS memory system where the battery is depleted or absent; in that case a boot occurs.

HALT: This position allows the use of the console keypad after power-up.

NOTE

To initialize the computer, depress the HALT/SI key while holding the START key down. The user should have the slide switch in the desired position, as the slide switch is examined during the initialization. This can be used to clear a hung bus without turning off power.

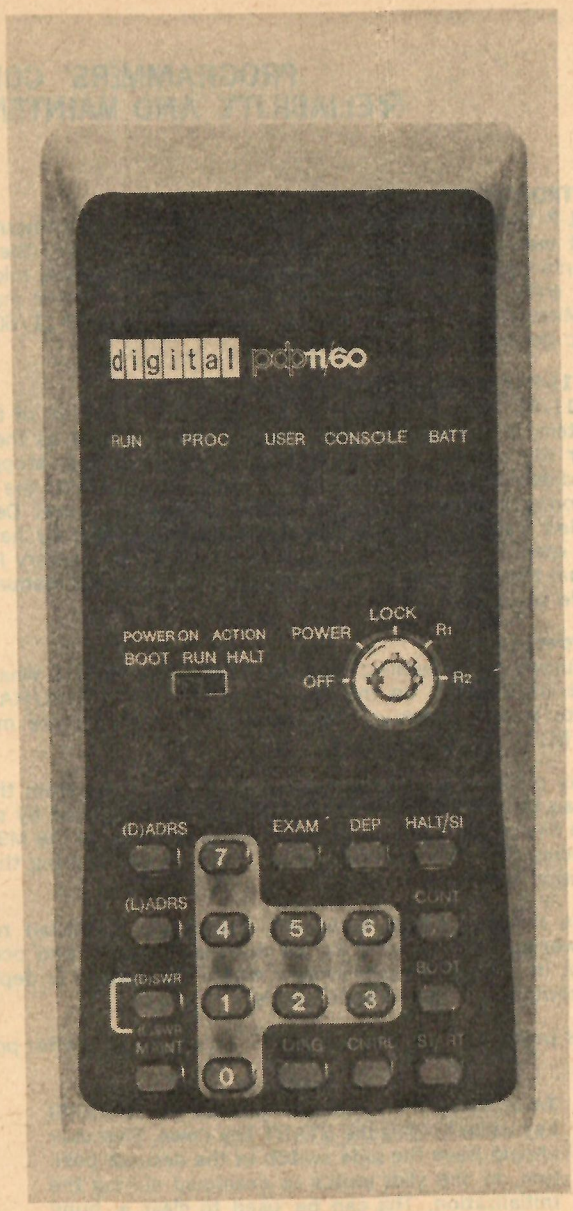


Figure 11-1 PDP-11/60 Console

11.2.2 Starting and Stopping

If it is desired to start a program from a given address, turn the power on after placing the slide switch in HALT position. The keypad is active and the desired address can be loaded into the Temporary Switch Register (and also in the display) by pressing the numeric switches. After checking the desired address as displayed, press the LADRS key. Then press START, holding CNTRL key down. This starts the program. The CONSOLE light goes out and the RUN light comes on (system is in RUN mode). The only keys which are active are the Numerics, DADRS, D/LSWR, and HALT/SI.

To terminate the execution of a program, depress the HALT/SI key. This stops the program, the CONSOLE light comes on and the RUN light goes out. The system is in CONSOLE mode and all the keys in the keypad are active. The display contains the PC. In this mode of operation, a single instruction is executed each time the HALT/SI key is depressed.

11.2.3 Console Indicators and Switches

The PDP-11/60 Programmers' Console provides the following facilities:

- Six-digit octal display for address and data indication.
- Processor state lights:
  RUN
  PROC (Processor)
  USER
  CONSOLE
  BATT (Battery)
- BOOT/RUN/HALT slide switch for power-up action.
- Five-position rotary switch for selection of machine status.
  OFF
  POWER
  LOCK (panel lock)
  R1 (Remote 1)
  R2 (Remote 2)
- Keypad switches (four rows of five switches each, noted below)
  DADRS (Display address)
  7 (Numeric)
  EXAM (Examine)
  DEP (Deposit)
  HALT/SI (Halt/Single Instruction)
  (L)ADRS (Load Address)
  4 (Numeric)
  5 (Numeric)
  6 (Numeric)
  CONT (Continue)
  (D)SWR, (L)SWR (Display Switch Register, Load Switch Register)
  1 (Numeric)
  2 (Numeric)
  3 (Numeric)
  BOOT (Bootstrap)
  MAINT (Maintenance)
  0 (Numeric)

DIAG (Diagnostic)
CNTRL (Control)
START

NOTE

The CNTRL interlocks the action of other keys. The functions labeled in blue on the control panel cause irrevocable change in machine status and therefore are interlocked with CNTRL. CNTRL must be depressed when the other key is activated for action to occur.

11.2.4 Console Internal Registers

The console has the following four internal registers (in the A and B Scratchpads) for its own exclusive use. Each is 16 bits wide and has the functions noted below:

CNSL.CNTL, Console Control, is a 16-bit register containing various control bits used in the console microcode. It also contains the upper two bits of the Temporary Switch Register, the Console Switch Register and Console Address Register.

CNSL.TMPSW, Console Temporary Switch Register, is 18 bits wide and is made up of the CNSL.TMPSW Register and two bits in the Control Register. The Temporary Switch Register is used as a buffer to collect the numerics and is also used for display.

CNSL.ADRS, Console Address Register, is also 18 bits wide and is composed of the CNSL.ADRS Register and two bits in the CNSL.CNTL to allow 18-bit physical addresses.

CNSL.SW, Console Switch Register, is also 18 bits wide and is composed of the CNSL.SW Register and two bits in the CNSL.CNTL Register. This register has a UNIBUS address of 777570 and is a read-only register. If a write is attempted at this address, the data will be written in the Console Address Register and then displayed on the console if the DLOCK bit in the CNSL.CNTL is not set. This bit is cleared in (D)ADRS and START functions and set in every other function. (D)ADRS can be used to unlock the display and provide a positive indication of movements by the program to 777570.

11.2.5 Functions of Switches and Indicators

Octal Display

The octal display is a six-digit, seven-segment display used to display address or data information. The display allows 18 bits (octally coded) to be displayed. For (L)ADRS, (D)ADRS, and HALT/SI functions, decimal points are displayed in conjunction with the octal address. For all other functions, the information is displayed without the decimal points.

Processor State Lights

RUN—If illuminated, indicates that the processor is executing instructions. The light will not remain illuminated during an extended WAIT instruction.

PROC—If illuminated, indicates that the processor is the master device and has control of the UNIBUS.

USER—If illuminated, indicates that the processor is in User mode and certain restrictions on instruction operation and Processor Status word (PS) loading exist.

CONSOLE—If illuminated, indicates that the processor is in Console mode and is under control of the console keypad switches (manual operation).

BATT—Battery monitor indicator. This indicator will function only in machines containing the battery backup option and has the following four states:

OFF—Indicates either no battery present, or battery depletion if battery is present.

ON (Continuous)—Indicates that battery is present and is charged.

Flashing (Slow)—Indicates battery is charging.

Flashing (Fast)—Indicates loss of power, and also that battery is discharging while maintaining MOS memory contents.

BOOT/RUN/HALT Slide Switch

Power-up action is determined by this switch position, in conjunction with PANEL LOCK status. If the rotary switch is in LOCK position (deactivating all keypad functions), inadvertent operation of the slide switch has no effect. Upon power-up, the slide switch is treated as if it were in the RUN position, regardless of its physical position. If the battery is depleted for a MOS memory system, RUN is altered to a BOOT action.

If the console is not in LOCK position, and a power-fail occurs, three choices of recovery (BOOT, RUN, and HALT) are available.

BOOT—Power-up to the M9301 bootstrap terminator.

RUN—Power up to location 24 which contains the power-up vector. Note that this action occurs independent of battery status on a MOS memory system.

HALT—Power-up to the console. The CONSOLE light is illuminated and the console keypad switches are active.

Rotary Switch

DC OFF—Removes dc power from processor and core memory (MOS memory battery charger is still on).

POWER—Applies power to all units. All console controls are operable in Console mode.

LOCK—Deactivates all keypad functions. With power switch in LOCK position, the position of the BOOT/RUN/HALT slide switch has no effect when power-up occurs; power-up is to RUN (unless a battery depletion causes BOOT upon a MOS memory system).

R1—Local control is deactivated to allow operation from a remote console. The octal display on the console will be blanked.

R2—Console action is the same as R1.

Keypad Switches

The keypad contains twenty switches which are priority-encoded into a unique five-bit code. Simultaneous operation of the keys will allow the

operation of the switch with the higher priority. The switches are listed in order of their priorities with the highest priority described first.

0-7NUMERICS—Activation of any of the numeric keys causes the binary value of that key to be entered into the low-order three bits of the Temporary Switch Register. The previous contents are left-shifted three bits. Each three-bit binary value is displayed in octal. For each additional numeric depressed, the Temporary Switch Register (one of four internal registers) is left-shifted three bits and the octal display is left-shifted one digit. Consequently, a six-digit octal number is generated as octal digits are entered from the right and left-shifted. Operation of the numerics occurs in both Console mode and Run mode.

NOTE

The CNTRL (Control) key is used in conjunction with some keys to prevent accidental operation of certain functions. When used, the CNTRL key must be depressed first and held down while the other key is depressed. Those keys which are interlocked with the CNTRL key are indicated with an asterisk.

HALT/SI (Halt/Single Instruction)—Depressing this switch while the processor is in RUN mode will halt the processor, between instructions, after outstanding trap sequences and before bus requests. The processor is now in CONSOLE mode and the CONSOLE indicator is lighted. The octal display indicates the program counter for both HALT and SI functions. Depressing the HALT/SI switch now causes a single instruction to be executed.

To initialize the system without a program start, it is necessary to depress the HALT/SI key while holding the START switch down.

NOTE

The PDP-11/60 differs from other PDP-11 processors regarding the single instruction step function. An operator cannot simply load an address and immediately start single-stepping. To start from an arbitrary address, the PC must be loaded using the Maintenance key function; one can then single-step by pressing the HALT/SI switch.

(D)SWR, *(L)SWR (Display Switch Register, Load Switch Register)—Displays the contents of the Console Switch Register in both CONSOLE and RUN modes. If this switch is depressed while the CNTRL switch is held, the contents of the Temporary Switch Register are loaded into the Console Switch Register. The contents of the Console Switch Register are displayed. Operative in both CONSOLE and RUN modes.

(D)ADRS—Displays the contents of the Console Adress Register and clears the DISPLAY LOCK bit, thus enabling the program movements to 777570. Operation occurs in both CONSOLE and RUN modes.

11.2.6 Console Mode Functions

Console operations are word-ordered operations. If an odd bus address

(bit 00 enabled) is used, the odd address is stored in the Console Address Register (CAR). Examine or deposit operations in this address will be treated as word operations (bit 00 ignored).

An EXAM or a DEP operation that references a non-existent address causes the machine to display the Console Address with all the decimal points lighted. Time-out trap sequences to non-existent addresses will not be activated.

NOTE

The following switches are active only in CONSOLE mode.

(L)ADRS (Load Address)—Depressing this switch transfers the contents of the Temporary Switch Register to the Console Address Register to be used in subsequent DEP or EXAM operations. The content of the Console Address Register is displayed in the octal display and all the decimal points are lighted.

EXAM (Examine)—Depressing this key accesses the UNIBUS address specified in the Console Address Register, and displays the contents of that address in the octal display. Sequential examines increment the address by two and display the contents of the incremented addresses. This incrementation process is stopped if any other key except numerics is pressed.

DEP (Deposit)—Depressing this switch deposits the contents of the Temporary Switch Register at the UNIBUS address specified by the Console Address Register. The Console Switch Register is not changed. To deposit data into sequential addresses, all that is necessary is to press the DEP key. This automatically word-increments the Console Address Register and deposits the data into the incremented address. This process is stopped if any key other than the numeric keys is depressed.

***CONT** (Continue)—Depressing this switch allows the processor to leave CONSOLE mode and continue operation at the present Program Counter (PC) location without a BUS INIT. The display is unaltered.

***START**—Depressing this switch begins machine operation at the address (PC) specified by the Console Address Register after a BUS INIT signal. Operation occurs only in CONSOLE mode and the CONSOLE mode light is turned off. The display is unaltered.

***BOOT** (Bootstrap)—Depressing this switch will cause a BUS INIT and will start the boot program of the M9301 module. The display is unaltered.

***DIAG** (Diagnostic)—Depressing this switch transfers control to the DCS (Diagnostic Control Store) module, if present. Otherwise, the computer enters CONSOLE mode. The display is unaltered.

MAINT (Maintenance)—This key is used to read and write the internal registers. The procedure for reading an internal register is:

*The functions with an asterisk are operative only when the CNTRL keypad switch is depressed.

1. Load the Temporary Switch Register with the read opcode of the register that is desired to be read. The opcodes for the internal registers are listed in Table 11-1.
2. Depress the (L)SWR keypad switch while holding the CNTRL keypad switch depressed. This transfers the contents of the Temporary Switch Register to the Console Switch Register.
3. Depress the MAINT keypad switch while holding the CNTRL switch depressed. The console display will display the contents of the register specified by the opcode in step 1.

The procedure for writing an internal register is:

1. Load the Temporary Switch Register with the write opcode of the register that it is desired to write. The opcodes are listed in Table 11-1.
2. Depress the (L)SWR keypad switch while holding the CNTRL keypad switch depressed. This transfers the contents of the Temporary Switch Register to the Console Switch Register.
3. Load the Temporary Switch Register with the data to be written by depressing the applicable numeric switches.
4. Depress the MAINT keypad switch while holding the CNTRL keypad switch depressed. The console display will display the data that has been written into the specified register.

NOTE

In Table 11-1, a register can have several names, depending upon its use at a given time. For example, in the C Scratchpad, the register with the read/write code of 100/300 can be used as a Floating Point (FP) Register or as the Log Jam Register.

TABLE 11-1 Internal Registers Read/Write Function Codes

ASP LO: A SCRATCHPAD [0:15]	
Register	Read/Write Code
R0	000/200
R1	001/201
R2	002/202
R3	003/203
R4	004/204
R5	005/205
R6	006/206
R7	007/207
FAC3[0]	010/210
FAC3[1]	011/211
FAC3[2]	012/212
FAC3[3]	013/213
FAC3[4]	014/214
FAC3[5]	015/215
USER R6	016/216
FDST3	017/217

Table 11-1 Internal Registers Read/Write Function Codes (cont.)

ASPHI: A SCRATCHPAD [16:31]	
Register	Read/Write Code
WCSA[0]	020/220
WCSA[1]	021/221
WCSADR	022/222
CNSL.CADR	023/223
R(SRC)	024/224
R(SRC X)	025/225
R(SRC I)	
R(T1A)	
R(VECT)	
R(DST)	026/226
R(DST X)	
R(T2A)	
R(DST I)	
CNSL. SW	
CNSL. TMPSW	027/227
FAC1[0]	030/230
FAC1[1]	031/231
FAC1[2]	032/232
FAC1[3]	033/233
FAC1[4]	034/234
FAC1[5]	035/235
GEN, WHAMI	036/236
FPSHI, FEC FDST 1	037/237

BSPLO: B SCRATCHPAD [0:15]	
Register	Read/Write Code
R0	040/240
R1	041/241
R2	042/242
R3	043/243
R4	044/244
R5	045/245
R6	046/246
R7	047/247
FAC2[0]	050/250
FAC2[1]	051/251
FAC2[2]	052/252
FAC2[3]	053/253
FAC2[4]	054/254
FAC2[5]	055/255
USER R6	056/256
FDST 2	057/257

Table 11-1 Internal Registers Read/Write Function Codes (cont.)

BSPHI: B SCRATCHPAD [16:31]	
Register	**Read/Write Code**
WCSB[0]	060/260
WCSB[1]	061/261
WCSB[2]	062/262
R(ZERO)	063/263
R(SRC)	064/264
R(SRC, X)	
R(ES)	
R(T1B)	
R(DST)	065/265
R(DST X)	
R(T2B)	
R(ES)	
R(IR)	066/266
CNSL.CNTL	067/267

CSP: C SCRATCHPAD [0:15]	
Registers	**Read/Write Code**
FP, LOG JAM	100/300
FP, LOG SERVICE	101/301
FP, LOG PBA	102/302
FP, LOG CUA	103/303
FP, LOG FLAG/INTR	104/304
FP, LOG WHAMI	105/305
FP, LOG CACHE DATA	106/306
FP, LOG TAG CPU	107/307
FP, CONSOLE	110/310
FP, CONSOLE	111/311
FP, CONSOLE	112/312
FP, CONSOLE	113/313
CONST 2	114/314
MD	115/315
CONST 0	116/316
CONST 1	117/317

Table 11-1 Internal Registers Read/Write Function Codes (cont.)

OTHER REGISTERS	
Registers	Read/Write Codes
JAM	140/— Read only
SERVICE	141/— Read only
PBA	142/— Read only
CUA	143/— Read only
FLAG	144/344
REV	146/—
DCS0	152/—
DCS1	153/—
D REG	—/345 Write only
S REG	—/346
COUNT	147/347
NUA	—/350
RES	—/351
INIT	—/352
	NO-OPS @
	340-343
	150-177
	120-137
	320-337
	352-377

11.3 INTRODUCTION—RELIABILITY AND MAINTENANCE

The significant maintenance feature of the PDP-11/60 is the availability of a wide spectrum of reliability and maintenance aids. This spectrum ranges from software (system, diagnostics, microdiagnostics) to hardware (parity, error status registers, microbreak, single clock). These aids are coordinated via the Reliability and Maintenance Program (RAMP).

RAMP is a corporate program spearheaded by Field Service, the main purpose being the development of trade-off data for use by DIGITAL's engineering groups in hardware design. Reliability means minimizing failures (for any reason) and maintainability means ease of maintenance and a minimum time spent in isolating faults and making repairs.

The design and packaging of the PDP-11/60 has placed great emphasis on RAMP. This means reduced mean time between failures (MTBF) and reduced mean time to repair (MTTR).

11.4 RELIABILITY

Reliability refers to the minimization of failures in hardware and software. Some hardware failures can be avoided through better cooling or lower stress upon components. In other instances, when failures do occur, it is important that the computer be less sensitive to the error (fault tolerant). The RAMP features are noted below with special emphasis upon fault tolerance.

11.4.1 Hardware Features

Partitioned Logic—This feature confines related logic to one single board. This aids in fault isolation, facilitates module snap-out, and reduces spare stock. Single boards are used for the memory management/cache, timing board, datapath, decode, microword and timing. This partitioning is important in conjunction with fault isolation of the Diagnostic Control Store (DCS) option and with processor specific macrodiagnostics.

High Memory Integrity—Proven cache and core memory or MOS memory with error correction code logic provide memory integrity. The high-speed cache memory is protected by parity on the data and tag fields. Diagnostic logic is present to diagnose failures in the parity logic.

Greater Memory Reliability—By operating core memory at below-maximum spec performance levels (at 1 μsec), greater memory reliability is obtained. (Performance is achieved by use of the cache.) MOS memory with error correction code provides up to 24,000 hours of MTBF.

UCS Data Integrity—UCS data integrity with complete parity protection (3 bits per control word) ensures that the customer can run his microcode (from the UCS) without corrupting the base machine or the operating system. Diagnostic logic is present to diagnose failures in the parity logic.

Status and Control Registers—These registers are contained in cache, UCS, main memory and memory management to provide error and status information and operational control. These registers are specifically useful to the macrodiagnostics.

UNIBUS Enhancements—UNIBUS enhancements with extra deskew time assure data stability and therefore, integrity of control and data information. Separate UNIBUS terminator allows UNIBUS margining in conjunction with UNIBUS margining equipment. The M9301-YH provides a basic test of UNIBUS and memory integrity during the bootloading of the system.

Console Enhancements—Static discharge is eliminated by an RC filter board which intercepts the console cable to the processor. This provides direct and positive protection against static discharge. Further indirect protection is provided by dual request to console before action is initiated.

Multilayer Design—Multilayer design of backplane and multilayer modules provide a regularity of connectivity which will reduce electrical noise problems.

Better Grounding—All external cabinet panels are grounded to eliminate and reduce personal hazards and intermittent failures. Grounds are easily identified with a green sticker and ground symbol.

Cable Separation—Separate power and signal cables reduce coupling and facilitate better servicing. Cable ducts are included to keep air passages free and avoid overheating.

DC Power Capacity—Separation of the power and logic units increases dc power capacity. Each H7420 power supply allows dc regulators to

power each logic card cage. A separate power supply is provided for the FP11-E floating point option.

Better Air Cooling System—Better air cooling is provided with prescribed routing for addons. Predetermined cable troughs eliminate interference with air flows, resulting in cooler, longer operation. The PDP-11S60 and PDP-11T60 avoid higher operating temperatures by obtaining room ambient rather than cabinet ambient air.

Safety—The solid top on the computer eliminates the possibility of liquids dropping into electronics (natural tendency is to store things on cabinet tops). UL/CSA approval confirms that safety hazards are eliminated.

Reduced Shipping Damage—Shipping damage due to shock and vibration is reduced in the PDP-11S60 and PDP-11T60 because of greater structural integrity. A caster system, featuring spring-loaded casters, is designed to absorb vertical shock and vibration when the lowboy cabinet is moved.

11.4.2 Fault Tolerance

Fault tolerance is designed to keep the system operating even when one component is generating random or sustained errors. There are certain fault conditions which are very difficult to simulate and observe in a short period of diagnostic testing.

Operating system software becomes a real-time error detection system. The fact that the system is encountering fault conditions is used to get useful information about the fault. Later on, in off-line mode, less time can be devoted to detection and resolution of fault. Thus down-time is reduced and system availability is increased.

A customer can write algorithms to forewarn of impending fatal failure by looking at the gradual deterioration of the system; this allows him to schedule his maintenance before the system actually fails.

Fault tolerance through dynamic reconfiguration is also possible using the error log as a performance index (e.g., shutting off banks of memory, dropping cache, dropping off-line faulty peripherals).

Confidence Check Features

The PDP-11/60 offers a confidence check by simple pushbutton operation using the M9301-YH bootstrap loader. In addition, parity is checked on the data during operation, and specific actions can be taken to minimize the effect of an error. Operating and error logging under RSX-11M system software allows a forewarning of impending fatal failures.

M9301-YH Diagnostic Bootstrap Loader

- Verifies that the system elements required to load the system monitor or diagnostic software are functioning properly (cache, processor, memory management and main memory).
- Exercises the instructions necessary to boot the system.
- Tests the memory area to be used by the secondary boot, which actually loads the system monitor or diagnostics.

- Provides diagnostic information to correct a ny problems.
- Tests are conducted automatically upon a boot request.

Base PDP-11/60 Power-Up Tests

- A check of the basic datapath hardware and microbranch instructions, executed whenever the processor initiates a power-up sequence through a microcode routine.
- An 8-step logic check which verifies that the processor's microcode can be executed, and that the internal datapaths are functional.

Parity Check During System Operation
While the system is in operation, extensive parity checking is performed; not only on data, but also on addresses. The PDP-11/60 employs one parity bit for every eight-bit byte (total 9 bits). When single-bit parity errors are detected, error information is recorded, and the system traps to a software subroutine. Byte parity checking is provided:

- For all memory—cache and core; on addresses and data in cache, on data in core
- Between the processor and cache (data)
- On Add, Data, and Control references to main memory
- UNIBUS to/from cache (data)

To ensure that all parity checkers are operating correctly, the software can force wrong parity on them.

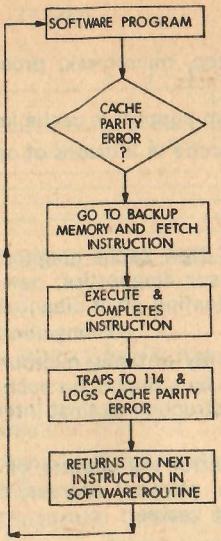
Fault Tolerance Through Error Logging
When parity errors do occur, the operating systems take corrective action by utilizing the hardware's parity error detection logic to isolate faulty memory and prevent its use.

This is a software technique for storing information about memory parity errors, processor-detected errors, and mass storage device errors in a history file.

After a parity error occurs (UCS, cache, or main memory) and executes a trap to location 114, it does a bit check to tell what kind of an error occurred. This allows the RSX-11M operating system to keep track of errors that occurred. DIGITAL system software handles soft errors or errors that can be recovered. Soft errors allow the system to continue.

The RSX-11M error log has two formats. The first is a summary table that shows what errors have occurred in the system; the second is an extraction of the error history that names the device on which the error occurred, gives the date and time of the error, the status registers and their contents, and the job number that was running when the error occurred. By examining the status register contents, it is possible to pinpoint exactly what happened when the error occurred. Information will vary according to the device and its status bits.

Fault Tolerance with Automatic Back-up on Cache Errors
When a cache parity error occurs, an automatic reference is made to the back-up memory and the information is fetched, executed and then a trap occurs to location 114 indicating a cache parity error.


SOFTWARE PROGRAM
CACHE PARITY ERROR ?
GO TO BACKUP MEMORY AND FETCH INSTRUCTION
EXECUTE & COMPLETES INSTRUCTION
TRAPS TO 114 & LOGS CACHE PARITY ERROR
RETURNS TO NEXT INSTRUCTION IN SOFTWARE ROUTINE

Fault Tolerance with Cache Turn-off
In the cache memory, hardware/software error correction combines to provide a failsoft memory system. The software can disable the cache and operate out of main memory if cache errors occur.

Fault Tolerance with Integral Floating Point Instruction Set
If there is a fault in the FP11-E floating point processor, the FP11-E can be turned off and the computer's basic floating point instructions can be used. System performance will be degraded, as the basic instructions are not executed as rapidly as the FP11-E instructions.

Fault Tolerance with User Control Store
A UCS parity error causes a trap to location 114. The PDP-11/60 has the ability to turn off UCS and continue with the base machine instruction set. Software routines can be substituted for the previously used microinstruction at degraded performance.

11.5 MAINTENANCE

Low mean time to repair (MTTR) is the main purpose for the maintenance aids in the PDP-11/60. These features are noted below.

Base System Integrity

- Six modules, separable with macrodiagnostics. High internal visibility with microbreak.
- Console isolation with static filter board.
- Console provides own microtest in the basic processor.
- Power supply has indicator lights and individual modular regulators.

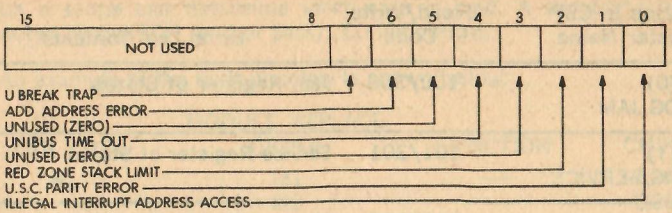
Tests for Module Isolation

- Hardware—Single microstep, microbreak, processor error log, single instruction.
- RSX-11M Software—System support to cache level.
- DCS Module—2K of microcode in 3 modes of operation:
  Manual
  Console
  External operating system
- Diagnostics without DCS—New cache diagnostics will support hardware features: new processor diagnostics, new memory management diagnostics, consolidated floating point diagnostics, cache diagnostics, UCS diagnostics.
- On-Line Diagnosis—Capability of using microbreak feature to execute parts of a macroinstruction by stopping on each microword and analyzing the word using MED instruction against internal registers and error log.
- Self-Diagnosis—Processor error log capture of information previously not available. UNIBUS data, physical address, cache, cache data, next microaddress, last interrupt vector.
- Internal Visibility—92 internal registers can be read by console or program (MED). It means access to processor error log or internal registers. Architecturally, it means macro/micro exchange capability that also has reliability/serviceability implications.
- Parity—The UCS, cache and memory are protected by parity. UCS errors are fatal, while cache and UNIBUS errors can allow degraded performance. Complete parity protection on cache against soft errors. Diagnostic aids to test and check the parity generators and checkers.
- M9301 Boot—The macro test programs of the M9301 are available to the machine upon use of the BOOT key switch, or power-up to BOOT mode.
- Processor Error Log—The machine logs significant error information into scratchpad registers. The information contained in the error log is that of the micro machine (UNIBUS also) at the time of error. This error log, in the first instance, can be read from the console or used by diagnostic programs. System software will use the CPU error and memory system error registers which are similar to that of the PDP-11/70 in format. See Table 11-2 for the address and contents of the processor error log.

NOTE

The PDP-11/60 Processor Error Log is created by the microcode at the point of error. The full error log can be created only if the error results in an abort of the instruction or bus cycle. If the error does not abort, the full error log is not created. For cache parity errors that go to backing store, sufficient information is stored to support the Memory System Error Register (Chapter 6).

CPU ERROR REGISTER 777766



11.5.1 PDP-11/60 Microdiagnostics

This is the first time the PDP-11 family has implemented microdiagnostics to give:

- Board-level fault resolution
- Trouble-shooting assistance and information to less than module level (chip or group of chips)
- Self-contained DCS (Diagnostic Control Store) module, no need for loading peripherals
- Execution time—milliseconds
- Confidence test of the CPU at the press of a button on the console
- Switch on DCS to provide self-checking of the DCS module
- Module-swap repair technique made possible, using less sophisticated personnel. Customer could call in the octal number to a Field Service engineer who could carry board with him.
- Low MTTR and higher availability.

11.5.2 KU116-BB Diagnostic Control Store

The PDP-11/60 Diagnostic Control Store (DCS) is the means by which microdiagnostics are stored and run in the 11/60 processor. This hex board contains a 2K x 52-bit ROM array, 12 indicator LEDs, four switches, control logic, and interface.

It is used in slot 1 in lieu of the UCS or ECS module. The DCS module contains its own diagnostic code in ROM and does not use the base machine code. It checks the basic processor modules (except the one board KT/cache which has its own macrodiagnostics).

The DCS module has three modes of access:

1. DIAGnose Switch on Console
   If the computer is in the halt position, the DIAGnose switch on the console initiates microdiagnostics. This provides a "confidence check" on the system and gives a go/no-go indication on the console by displaying a defined octal number in numeric form.
2. Switch on DCS Module
   A switch on the DCS module can be used to initiate the microdiagnostics in the ROM pattern on the DCS. An error is indicated by an er-

Table 11-2 Processor Error Log

Location in CSP Register Name	Read/Write Code	Error Log Contents
CSP(0) LOG.JAM	100/300	Jam Register of Status
CSP(1) LOG.SERVICE	101/301	Service Register of Status
CSP(2) LOG.PBA	102/302	Physical Bus Address Register (Bits 17, 16 in Service Register Log)
CSP(3) LOG.CUA	103/303	Microprogram Address
CSP(4) LOG.FLAG/INTR	104/304	Flag Request Register of Status Last Vector Serviced.
CSP(5) LOG.WHAMI	105/305	Variety of Processor Status Bits, Processor Option Status.
CSP(6) LOG.CACHEDATA	106/306	Cache Data
CSP(7) LOG.TAG/HITR	107/307	Cache Tag Data Hit Register
CSP(10) CNSL.CNST100000	110/310	
CSP(11) LOG.DS	111/311	Data Stored
CSP(12) CNSL.CNST30000	112/312	
CSP(13) CNSL.CNST040000	113/313	
CSP(14)	114/314	
CSP(15)	115/315	
CSP(16)	116/316	
CSP(17)	117/317	

ror light (LED) which also displays an octal code in 12 LEDs on the DCS module. By consulting the fault directory, you can determine which module is bad and do a module replacement. Also, if a fault is detected, the routine automatically goes to an error loop on micro-

test, allowing trouble shooting to the chip level. A Field Service technician's interpretation of the fault directory and signals he receives on a scope can determine which chip is bad. A successful test is noted by the EOP (end of pass) LED indication.

An example of the fault directory is shown below:

ERR CODE	MODULE REPLACE ORDER	FUNCTION	CHIPS
N_8	M1	—	E24
	M2		E26

3. MED Instruction Access
   A Maintenance, Examine and Deposit instruction is provided. This instruction allows macroinstruction access to the internal (or micro) state of the machine to monitor, examine, deposit internal flags, registers, etc., of the machine at the program level. The main system use of this instruction is to read and reset the error log. External system use of the DCS module requires a separate computer to start and read error indicators.

11.5.3 Maintenance Features of the System Package

Better Serviceability

- Card cage swings out for servicing
- Console is not in front of logic
- Visibility of processor registers through LEDs
- System units with distribution board for power
- Power supplies are mounted in back, separate from logic

Box Separation

The separation of the logic cage from the power supplies provides an independence of power. The modular regulators used are for different voltages, and can be identified as failed and then replaced with ease.

Physical Enhancements

- Cables.
- Non-sliding card cage reduces service loops.
- Steel-comb on card cage eases module removal/insertion with cables attached.
- No module cables exist on the processor modules. Only a single 40-wire console cable exists and comes from the backplane.
- Cabinet comes with cable troughs.
- Prescribed routing reduces electrical noise or crosstalk. Uniform wiring patterns for duplicate systems. Looks neater; one system looks like the next; easier for Field Service.
- Easier module insertion. Continuous card guides are color-coded alternately red and white. Better module alignment. Less connection damage. Easier to locate the slot.

11.5.4 Maintenance Features of the Console

Console Microtest in Processor Microcode
This feature is permanently resident in the base machine ROM and allows checking and repair of the console even with the base machine down.

Console Enhancements
Remote Console—A rotary switch setting and a backpanel connector (20 pin) is provided to support a remote console interface.

Maintenance, Exam and Deposit of Internal Registers—Error flags and control bits in this microprocessor are treated as data. 92 registers contain internal information and can be directly accessed by the console.

Numerical Display—The keypad and display provide an octal number to conform with diagnostic documentation.

11.5.5 Maintenance Through Software and Firmware

Software Enhancements

- DCS Microdiagnostics—Diagnostic program in ROM allows Field Service to isolate faults to the module level.
- Macrodiagnostics—News diagnostics for CPU, cache, UCS and memory management are available, and provide a higher degree of coverage and resolution. Customer-useable for routine checking to provide high operational reliability. Low MTTR and down-time.
- Error Handling Diagnostics—This diagnostic tool has been added to DEC X-11 Diagnostic Package to check system interaction. This provides a capability to exercise complete system and diagnose system faults, in some cases, to a module level.
- RSX-11M System Software—Historical data between PMs from the error log, CPU Error Register and (Memory) System Error Register increases 'on' time and reduces the number of service calls.

Micromachine Access

- Microbreak—Pulse produced at microbreak location which can be used to trigger service scopes.
- Break mode where flow is interrupted and machine goes to console with an error log. In the console, the internal machine and error log can be examined.
- Microstep Capability—Single clock steps; single microstep operation can be performed with LEDs on NUA (next micro address).
- Single instruction operation can be performed at the console.
- Microdiagnostics—It allows error analysis even if machine is completely inoperable (including console).

APPENDIX A

UNIBUS ADDRESSES

A.1 INTERRUPT & TRAP VECTORS

000	(reserved)
004	Illegal instructions, Bus Errors, Stack Limit, Illegal Internal Address, Microbreak. Microbreak.
010	Reserved instructions
014	BPT, breakpoint trap (Trace)
020	IOT, input/output trap
024	Power Fail
030	EMT, emulator trap
034	TRAP instruction
040	System software
044	System software
050	System software
054	System software
060	Console Terminal, keyboard/reader
064	Console Terminal, printer/punch
070	PC11, paper tape reader
074	PC11, paper tape punch
100	KW11-L, line clock
104	KW11-P, programmable clock
110	
114	Memory system errors (Cache, UNIBUS Memory, UCS Parity)
120	XY Plotter
124	DR11-B DMA interface; (DA11-B)
130	ADO1, A/D subsystem
134	AFC11, analog subsystem
140	AA11, display
144	AA11, light pen
150	
154	
160	
164	
170	User reserved
174	User reserved
200	LP11/LS11, line printer
204	RS04/RF11, fixed head disk
210	RC11, disk
214	TC11, DECtape
220	RK11, disk
224	TU16/TM11, magnetic tape
230	CD11/CM11/CR11, card reader
234	UDC11, digital control subsystem; ICS/ICR11
240	PIRQ, Program Interrupt Request (11/55,11/45)

244 Floating Point Error
250 Memory Management
254 RP04/RP11 disk pack
260 TA11, cassette
264 RX11, floppy disk

270 User reserved
274 User reserved

300 (start of floating vectors)

A.2 FLOATING VECTORS

There is a floating vector convention used for communications (and other) devices that interface with the PDP-11. These vector addresses are assigned in order starting at 300 and proceeding upwards to 777. The following Table shows the assigned sequence. It can be seen that the first vector address, 300, is assigned to the first DC11 in the system. If another DC11 is used, it would then be assigned vector address 310, etc. When the vector addresses have been assigned for all the DC11's (up to a maximum of 32), addresses are then assigned consecutively to each unit of the next highest-ranked device (KL11 or DP11 or DM11, etc.), then to the other devices in accordance with the priority ranking.

Priority Ranking for Floating Vectors

(starting at 300 and proceeding upwards)

Rank	Device	Vector Size (in octal)	Max No.
1	DC11	$(10)_8$	32
2	KL11, DL11-A, DL11-B	10	16
3	DP11	10	32
4	DM11-A	10	16
5	DN11	4	16
6	DM11-BB (DH11-AD or DV11)	4	16
7	DR11-A	10*	32
8	DR11-C	10*	32
9	PA611 Reader	4*	16
10	PA611 Punch	4*	16
11	DT11	10*	8
12	DX11	10*	4
13	DL11-C, DL11-D, DL11-E	10	31
14	DJ11	10	16
15	DH11	10	16
16	GT40	10	1
17	LPS11	30*	1
18	DQ11	10	16
19	KW11-W	10	1
20	DU11	10	16
21	DUP11	10	
22	DV11	10	

*—The first vector for the first device of this type must always be on a $(10)_8$ boundary.

A.3 FLOATING ADDRESSES

There is a floating address convention used for communications (and other) devices interfacing with the PDP-11. These addresses are assigned in order starting at 760 010 and proceeding upwards to 763 776.

Floating addresses are assigned in the following sequence:

Rank	Device
1	DJ11
2	DH11
3	DQ11
4	DU11

A.4 DEVICE ADDRESSES

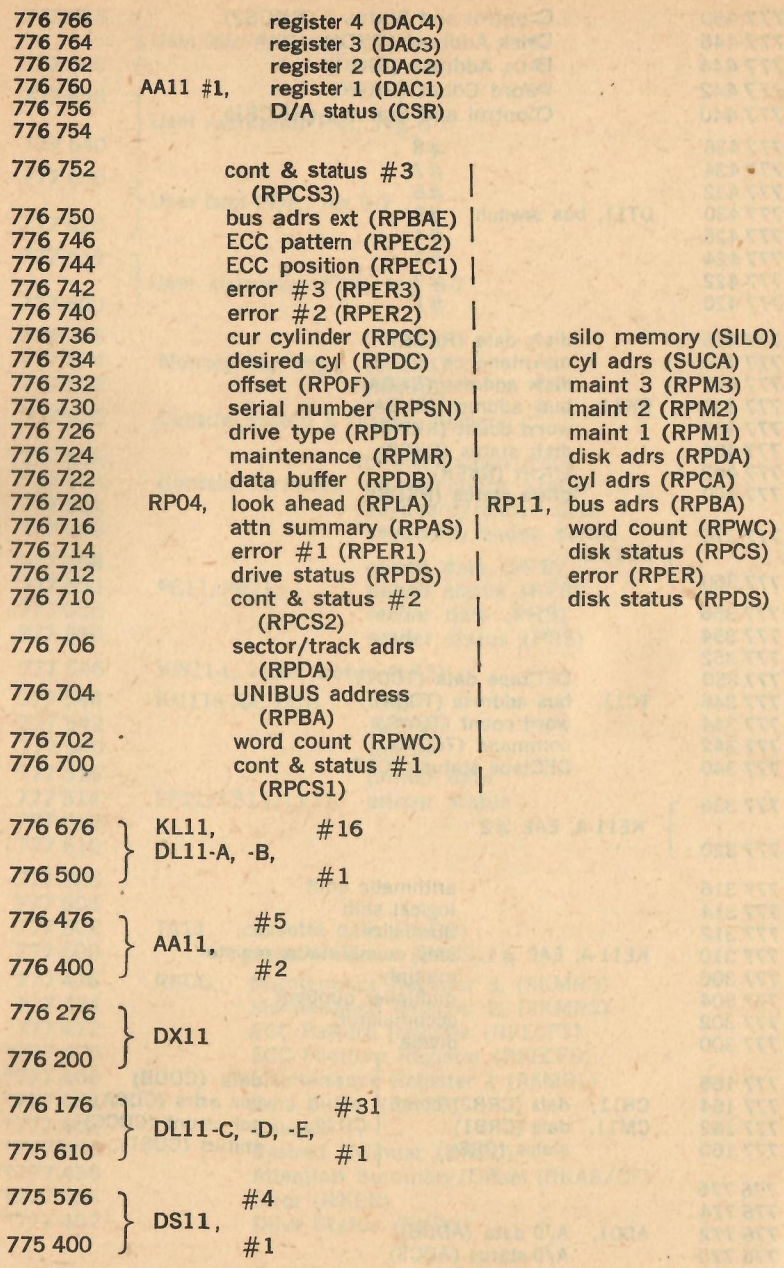
777 776	Processor Status word (PS)	
777 774	Stack Limit (SL)	
777 772	Program Interrupt Request (PIR)	
777 770	Microprogram Break	
777 766	CPU Error	
777 764	System I/D	
777 762	Upper Size } System Size	
777 760	Lower Size }	
777 756		
777 754		
777 752	Hit/Miss	
777 750	Maintenance	
777 746	Cache Control	
777 744	Memory System Error	
777 742	High Error Address	
777 740	Low Error Address	
777 717	User	R6 (SP)
777 716	Supervisor	R6 (SP)
777 715		R5
777 714		R4
777 713	General registers,	R3
777 712	Set 1	R2
777 711		R1
777 710		R0
777 707		R7 (PC)
777 706	Kernel	R6 (SP)
777 705		R5
777 704		R4
777 703	General registers,	R3
777 702	Set 0	R2
777 701		R1
777 700		R0

777 676 777 660	User Data PAR, reg 0-7
777 656 777 640	User Instruction PAR, reg 0-7
777 636 777 620	User Data PDR, reg 0-7
777 616 777 600	User Instruction PDR, reg 0-7
777 576	(MMR2)
777 574	Memory Mgt regs, (MMR1)
777 572	(MMR0)
777 570	Console Switch & Display Register
777 566	printer/punch data
777 564	Console Terminal, printer/punch status
777 562	keyboard/reader data
777 560	keyboard/reader status
777 556	punch data (PPB)
777 554	PC11/PR11, punch status (PPS)
777 552	reader data (PRB)
777 550	reader status (PRS)
777 546	KW11-L, clock status (LKS)
777 544	KU116-AA, UCS Data
777 542	Address
777 540	Status
777 516	printer data
777 514	LP11/LS11/LV11, printer status
777 512	
777 510	
777 506	
777 504	
777 502	TA11, cassette data (TADB)
777 500	cassette status (TACS)
777 476	RK06, Maintenance Register 3, (RKMR3)
777 474	Maintenance Register 2, (RKMR2)
777 472	ECC Pattern Register (RKECPT)
*777 470	ECC Position Register (RKECPS)
*777 466	Maintenance Register 1 (RKMR1)
*777 464	Data Buffer (RKDB)
*777 462	Unused
*777 460	Desired Cylinder (RKDC)
*777 456	Attention Summary/Offset (RKAS/OF)
*777 454	Error (RKER)
*777 452	Drive Status (RKDS)

*Also used by RF 11

**777 450		Control and Status 2 (RKCS2)
**777 446		Disk Address (RKDA)
**777 444		Bus Address (RKBA)
*777 442		Word Count (RKWC)
**777 440		Control and Status 1 (RKCS1)
777 436		#8
777 434		#7
777 432		#6
777 430	DT11, bus switch	#5
777 426		#4
777 424		#3
777 422		#2
777 420		#1
777 416		disk data (RKDB)
777 414		maintenance
777 412		disk address (RKDA)
777 410	RK11,	bus address (RKBA)
777 406		word count (RKWC)
777 404		disk status (RKCS)
777 402		errorr (RKER)
777 400		drive status (RKDS)
777 376	} DC14-D	
777 360		
777 356		
777 354		
777 352		
777 350		DECtape data (TCDT)
777 346	TC11,	bus address (TCBA)
777 344		word count (TCWC)
777 342		command (TCCM)
777 340		DECtape status (TCST)
777 336	} KE11-A, EAE #2	
777 320		
777 316		arithmetic shift
777 314		logical shift
777 312		normalize
777 310	KE11-A, EAE #1,	step count/status register
777 306		multiply
777 304		multiplier quotient
777 302		accumulator
777 300		divide
777 166		data (CDDB)
777 164	CR11/ data (CRB2) comp	cur adrs (CDBA)
777 162	CM11, data (CRB1)	CD11, col count (CDCC)
777 160	status (CRS)	status (CDST)
776 776		
776 774		
776 772	ADO1,	A/D data (ADDB)
776 770		A/D status (ADCS)

**Also used by RC 11


776 766
register 4 (DAC4)
776 764
register 3 (DAC3)
776 762
register 2 (DAC2)
776 760
AA11 #1,
register 1 (DAC1)
776 756
D/A status (CSR)
776 754
776 752
cont & status #3 (RPCS3)
776 750
bus adrs ext (RPBAE)
776 746
ECC pattern (RPEC2)
776 744
ECC position (RPEC1)
776 742
error #3 (RPER3)
776 740
error #2 (RPER2)
776 736
cur cylinder (RPCC)
silo memory (SILO)
776 734
desired cyl (RPDC)
cyl adrs (SUCA)
776 732
offset (RPOF)
maint 3 (RPM3)
776 730
serial number (RPSN)
maint 2 (RPM2)
776 726
drive type (RPDT)
maint 1 (RPM1)
776 724
maintenance (RPMR)
disk adrs (RPDA)
776 722
data buffer (RPDB)
cyl adrs (RPCA)
776 720
RP04,
look ahead (RPLA)
RP11,
bus adrs (RPBA)
776 716
attn summary (RPAS)
word count (RPWC)
776 714
error #1 (RPER1)
disk status (RPCS)
776 712
drive status (RPDS)
error (RPER)
776 710
cont & status #2 (RPCS2)
disk status (RPDS)
776 706
sector/track adrs (RPDA)
776 704
UNIBUS address (RPBA)
776 702
word count (RPWC)
776 700
cont & status #1 (RPCS1)
776 676
KL11,
#16
DL11-A, -B,
776 500
#1
776 476
#5
AA11,
776 400
#2
776 276
DX11
776 200
776 176
#31
DL11-C, -D, -E,
775 610
#1
775 576
#4
DS11,
775 400
#1

775 376	DN11,	#16	
775 200		#1	
775 176	DM11,	#16	DV11, #1-4
775 000		#1	
774 776	DP11,	#1	
774 400		#32	
774 376	DC11,	#32	
774 000		#1	
773 766	BM792, BM873 ROM PDP-11 diagnostic bootstrap (half of it)		
773 000			
772 776	PA611 typeset punch		
772 700			
772-676	PA611 typeset reader		
772 600			
772 576		maintenance (AFMR)	
772 574	AFC11,	MX channel/gain (AFCG)	
772 572		flying cap data (AFBR)	
772 570		flying cap status (AFCS)	
772 556	XY11 plotter		
772 550			
772 546			
772 544		counter	
772 542	KW11-P,	count set	
772 540		clock status	
772 536			
772 534			
772 532		read lines (MTRD)	
772 530		tape data (MTD)	
772 526	TM11,	memory address (MTCMA)	
772 524		byte record counter (MTBRC)	
772 522		command (MTC)	
772 500		tape status (MTS)	
772 516	Memory Mgt reg (MMR3)		
772 476		cont & status #3 (MTCS3)	
772 474		bus adrs ext (MTBAE)	
772 472		tape control (MTTC)	
772 470		serial number (MTSN)	

772 466		drive type (MTDT)
772 464		maintenance (MTMR)
772 462		data buffer (MTDB)
772 460		check character (MTCK)
772 456	TU16,	attention summary (MTAS)
772 454		error (MTER)
772 452		drive status (MTDS)
772 450		cont & status #2 (MTCS2)
772 446		frame count (MTFC)
772 444		UNIBUS address (MTBA)
772 442		word count (MTWC)
772 440		cont & status #1 (MTCS1)
772 436	} DR11-B #2	
772 430		
772 416		data (DRDB)
772 414	DR11-B #1,	status (DRST)
772 412		bus address (DRBA)
772 410		word count (DRWC)
772 376	} Kernel Data PAR, reg 0-7	
772 360		
772 356	} Kernel Instruction PAR, reg 0-7	
772 340		
772 336	} Kernel Data PDR, reg 0-7	
772 320		
772 316	} Kernel Instruction PDR, reg 0-7	
772 300		
772 276	} Supervisor Data PAR, reg 0-7	
772 260		
772 256	} Supervisor Instruction PAR, reg 0-7	
772 240		
772 236	} Supervisor Data Descriptor PDR, reg 0-7	
772 220		
772 216	} Supervisor Instruction Descriptor PDR, reg 0-7	
772 200		
772 136	} UNIBUS Memory Parity	
772 110		

```
772 072          cont & status #3 (RSCS3)
772 070          bus adrs ext (RSBAE)
772 066          drive type (RSDT)
772 064          maintenance (RSMR)
772 062          data buffer (RSDB)
772 060          look ahead (RSLA)
772 056          attention summary (RSAS)
772 054   RS04,  error (RSER)
772 052          drive status (RSDS)
772 050          control & status #2 (RSCS2)
772 046   RS04,  desired disk adrs (RSDA)
772 044          UNIBUS address (RSBA)
772 042          word count (RSWC)
772 040          control & status #1 (RSCS1)

772 016 }
        } GT40 #2
772 010 }

772 006          Y axis
772 004          X axis
772 002   GT40 #1 status
772 000          program counter

771 776          status (UDCS) |
771 774   UDC11,  scan (UDSR)  | ICS/ICR11
771 772                        |
771 770

771 776 }
        } UDC functional I/O modules
771 000 }

770 776 }        #8
        } KG11,
770 700 }        #1

770 676 }          #16
        } DM11-BB,
770 500 }          #1

770 436          DMA
770 434
770 432
770 430
770 426
770 424
770 422          ext DAC
770 420          D/A YR
770 416          D/A XR
770 414          D/A SR
770 412   LPS11,  D I/O output
770 410          D I/O input
770 406          CKBR
770 404          CKSR
770 402          ADBR
770 400          ADSR
```

770 366 770 200	UNIBUS Map	
767 776 766 000	GT40 bootstrap	
765 776 765 000	PDP-11 diagnostic bootstrap (half of it)	User & Special Systems
763 776	(top of floating addresses)	
760 010	(start of floating addresses)	

NOTE

All presently unused UNIBUS addresses are reserved by Digital.

APPENDIX B

PDP-11/60 INSTRUCTION TIMING

B.1 PDP-11/60 INSTRUCTION EXECUTION TIME

The execution time for an instruction depends on the instruction itself, the modes of addressing used, and the type of memory being referenced. In the most general case, the Instruction Execution Time is the sum of a Source Address Time, a Destination Address Time, and an Execute, Fetch Time.

Instr Time = SRC Time + DST Time + EF Time

Some of the instructions require only some of these times and are so noted. Times are typical and are based upon the MM11-WP memory as backing store. The simplified presentation of the timing data has occasionally resulted in a larger time for an instruction being noted. All times may vary +10% due to clock and bus tolerances.

B.2 BASIC INSTRUCTION SET TIMING

Double Operand

all instructions,
except MOV: Instr Time = SRC Time + DST Time + EF Time
MOV: Instr Time = SRC Time + EF Time (word only)

Single Operand

all instructions: Instr Time = DST Time + EF Time or
Instr Time = SRC Time + EF Time

Branch, Jump, Control, Trap & Misc

all instructions: Instr Time = EF Time

EIS (MUL, DIV, ASH, ASHC)

all instructions: Instr Time = DST Time + EF Time

Floating Point

all instructions:
except ABSF, ABSD,
NEGF, and NEGD: Instr Time = SRC Time + EF Time
ABSF, ABSD,
NEGF and NEGD: Instr Time = DST Time + EF Time

B.2.1 Using the Chart Times

To compute a particular instruction time, first find the instruction "EF" Time. Select the proper EF Time for the SRC and DST modes. Observe all "NOTES" to the EF Time by adding the correct amount to basic EF number.

Next, note whether the particular instruction requires the inclusion of SRC and DST Times; if so, add the appropriate amounts to correct EF number.

B.2.2 Chart Times

The times given in the chart are for Cache "hits"; that is, all the read cycles are assumed to be in the Cache. The number of read cycles in each subset of the instruction is also included so that timing can be calculated for a specific case of hits and misses, or timing can be calculated based on an average hit rate.

a) Specific hits and misses
Add 1.1 μsec for each read cycle which is a miss instead of a hit.

b) Average hit rate
If P_H is the percent of reads that are hits, add $1.1 \times (1 - P_H) \times$ (number of read cycles) to the instruction timing.

For example, an ADD A,B instruction using Mode 6 (indexed) address modes:

1) All Hits:

SRC time	= 0.85 μsec	2 read cycles
DST time	= 0.85 μsec	2 read cycles
EF time	= 2.2 μsec	1 read cycle
TOTAL	= 3.9 μsec	5 read cycles

2) 4 Hits, 1 Miss
Total = 3.9 + 1.1
= 5.0 μsec.

3) Read hit rate of 87%
Total = 3.9 + (1.1)(1 − .87)(5)
= 4.6 μsec.

NOTES

1. The times specified generally apply to Word instructions. In most cases Even Byte instructions have the same time, with some Odd Byte instructions taking longer. All exceptions are noted.
2. Timing is given without regard for NPR or BR serving.
3. Times are not affected if Memory Management is enabled.
4. All times are in microseconds, except where noted.

Table B-1 Source Address Time

Instruction	Source Mode	SRC Time	Read Memory Cycle
	0	.00	0
	1	.51	1
	2	.51	1
Double	3	1.0	2
Operand	4	.68	1
	5	1.2	2
	6	.85	2
	7	1.4	3

Table B-2 Destination Address Time

Instruction	DST Mode	DST Time (A)	Read Memory Cycle
Single Operand and Double Operand (except MOV, MTPI, MTPD, JMP, JRS)	0	.00	0
	1	.51	1
	2	.51	1
	3	1.0	2
	4	.68	1
	5	1.2	2
	6	.85	2
	7	1.4	3

NOTE (A): Add .17 μsec for odd byte instructions, except DST Mode 0.

Table B-3 Execute, Fetch Time
(Double Operand)

Instruction (Use with SRC Time and DST Time)	EF Time (SRC Mode 0) (DST Mode 0)	Read Mem. CYC	EF Time (SRC Mode 1-7) (DST Mode 0)	Read Mem. CYC	EF Time (SRC Mode 0-7) (DST Mode 1-7)	Read Mem. CYC
ADD, SUB, BIC, BIS	.34	1	1.0	1	2.2	1
CMP, BIT	.34	1	1.0	1	1.0	1
XOR	.34	1	—	—	1.0	1
MOVB	.34	1	.51	1	.51	1

Instruction (Use with SRC Time)	DST Mode	DST Register	EF Time (SRC Mode = 0)	EF Time (SRC Mode 1-7)	Read Memory Cycle
MOV	0	0-7	.34	.51	1
	1	0-7	1.0	1.0	1
	2	0-7	1.0	1.0	1
	3	0-7	1.4	1.4	2
	4	0-7	1.2	1.0	1
	5	0-7	1.5	1.5	2
	6	0-7	1.2	1.4	2
	7	0-7	1.7	1.9	3

Table B-4 Execute, Fetch Time
(Single Operand)

Instruction (Use with DST Time)	EF Time (DST Mode = 0)	Read Memory Cycle	EF Time (DST Mode 1-7)	Read Memory Cycle
TST	.34	1	.68	1
CLR, COM, INC, DEC, ADC, ROL, ASL	.34	1	1.9	1
NEG, SBC, ROR, ASR	1.2	1	2.4	1

Instruction	EF Time	Read Memory Cycle	
MFPI, MFPD	6.1	1	Use with SRC Times

Instruction	DST Mode	Instruction Time	Read Memory Cycle
MTPI, MTPD	0	3.6	1
	1	6.1	2
	2	6.3	2
	3	6.6	3
	4	6.3	2
	5	6.8	3
	6	6.6	3
	7	7.1	4

Table B-5 Branch Instructions

Instruction	Instruction Time	Read Memory Cycle
BR, BNE, BEQ, BPL, BMI, BVC, BVS, BCC, BGS, BGE, BLT, BGT, BLE, BHI, BLOS, BHIS, BLO	.85	1
SOB	2.0	1

Table B-6 JUMP Instructions

Instruction	DST Mode	Instruction Time	Read Memory Cycle
JMP	1	1.2	1
	2	1.4	1
	3	1.5	2
	4	1.4	1
	5	1.7	2
	6	1.4	2
	7	1.9	3

Instruction	DST Mode	Instruction Time	Read Memory Cycle
JSR	1	2.5	1
	2	2.7	1
	3	2.9	2
	4	2.7	1
	5	3.2	2
	6	2.9	2
	7	3.6	3

Table B-7 Miscellaneous Instructions

Instruction	Instruction Time	Read Memory Cycle
RTS	1.5	2
MARK	2.4	2
RTI	2.4	3
RTT	3.1	3
SET N, V, Z, C	1.5	1
CLR N, V, Z, C		
RESET	10 msec	1
IOT, EMT, BPT, TRAP	4.6	3

Table B-8 EIS Instructions MUL, DIV, ASH, ASHC

Source Address Time

Source Mode	Time (μsec)	Read Memory Cycle
0	.340	0
1	.640	1
2	.640	1
3	1.19	2
4	.85	1
5	1.36	2
6	1.19	2
7	1.70	3

Add 1.1 μsec for each read cycle which is a miss

EF Time

Instruction	EF Time (All Modes)	Read Memory Cycle
DIV	7.65 μsec	1
MUL	6.12 μsec	1
*ASH	3.57 μsec	1
*ASHC	4.25 μsec	1

*Add .17 μsec for each shift

B.3 FLOATING POINT INSTRUCTION TIMING

Floating point instruction times are calculated in a manner similar to the calculation of CPU instruction timing. However, due to the fact that the FP11-E is a separate processor, calculation of floating point instruction times must take this parallel or independent processing into account. See Table B-9 for definitions of some of the terms used in this appendix.

The following paragraphs provide a description of the method used to calculate effective instruction execution times.

NOTE

Resync and Interaction Times are not considered since handshaking synchronization overhead has been eliminated by the use of decoding and instruction fetch logic. That is, instruction fetch of floating point is initiated by the CPU but is received simultaneously by both processors.

In addition to instruction fetch and address calculation, the CPU converts fixed to floating point notation and, in some instances, fully executes the instruction (for example, LDFPS).

Table B-9 Definitions

Term	Definition
Instruction Decode	CPU time required to decode a floating point instruction op code. This time is fixed at 340 nsec.
Address Calculation Time	CPU time required to calculate the address of the operand. This time is dependent on the addressing mode specified. Refer to Tables B-11 and B-12.
Wait Time	CPU time spent waiting for completion by the floating point processor of a previous floating point instruction in the case of a load class of instruction. For store class instructions, the wait time is the summation of time during which the floating point processor completes a previous floating point instruction and floating point execution time for store class instruction. Wait time is calculated as follows: LOAD CLASS INSTRUCTIONS Wait Time = [Floating point execution (previous FP instruction)]—[Disengage and fetch time]—[CPU execution time for interposing non-floating point instruction]—[Instruction fetch time]—[Address calculation time]. If the result is ≤ 0, the wait time is 0. STORE CLASS INSTRUCTIONS Wait Time = { [Floating point execution time (previous FP instruction)]—[Disengage and fetch time]—[CPU execution time for interposing non-floating point instruction]—[Instruction fetch time] } * + [Floating point execution time]—[Address calculation time]. If the result is ≤ 0, the wait time is 0. *If the result of calculation { } is ≤ 0, then it becomes 0.
Argument Transfer Time	CPU time required to fetch and transfer operands. This time is 340 nsec $\times$ the number of 16-bit words read from memory or 1170 nsec $\times$ the number of 16-bit words written into memory. Add 1.075 μsec for a word received from memory that is a miss.
Shared Execution Time	CPU time spent in the execution of integer convert routines or any one of the instructions in category 5. Refer to Table B-13.

Table B-9 Definitions (Cont.)

Term	Definition
Disengage and Fetch	Time required to fetch the next instruction from memory. This time is fixed at 340 nsec for a cache hit. Add 1.075 μsec for a cache miss (MM110).
Floating Point Execution Time	Time required by the floating point processor to complete a floating point instruction once it has received all operands (load class). Refer to Table B-14.
Effective Execution	Total CPU time required to execute a floating point instruction.
	Effective execution time = instruction decode + address calculation + wait time + argument transfer time + shared execution time + disengage and fetch.

The floating point instruction set is divided into 5 categories as shown in Table B-10.

Table B-10 Floating Point Instructions

Category	Instruction
LOAD CLASS	LDF, LDD ADF, ADD SUBF, SUBD MULF, MULD DIVF, DIVD MODF, MODD LDCF, LDCD CMPF, CMPD LDCIF, LDCID LDCLF, LDCLD
LOAD CLASS (INTEGER CONVERT)	LDEXP
STORE CLASS	STF, STD STCDF STCFD
STORE CLASS (INTEGER CONVERT)	STCFI STCFL STCDI STCDL STEXP

Table B-10 Floating Point Instructions (Cont.)

Category	Instruction	
NULL CLASS (CPU EXECUTES)	CLRF, CLRD	Not M0
	NEGF, NEGD	Not M0
	ABSF, ABSD	Not M0
	TSTF, TSTD	Not M0
	LDFPS	
	STFPS	
	STST	
	CFCC	
	SETF, SETD	
	SETI, SETL	

Table B-11 Address Calculation (Floating/Double)

Mode	Time (nsec)	Read Memory Cycle
0	0	0
1	510	0
2	510	0
3	850	1
4	850	0
5	1360	1
6	850	1
7	1360	2

Table B-12 Address Calculation (Integer)

Mode	Time (nsec)	Read Memory Cycle
0	340	0
1	340	0
2	340	0
3	850	1
4	510	0
5	1020	1
6	850	1
7	1360	2

Table B-13 Shared Execution Time

Instruction		Time (nsec)
1.	CLRF CLRD (Not M0)	2210 2720
2.	NCGF NCGD (Not M0)	3060 3400
3.	ABSF ABSD (Not M0)	3060 3400
4.	TSTF TSTD (Not M0)	3060 3400
5.	LDFPS STFPS STST	2040 1360 2550
6.	CFCC SETF SETI SETD SETL	1020 1190 1360 1190 1360
7.	STEXP	2210
8.	LDEXP	1700

Table B-14 FP11-E Execution Times (nsec)

Instruction		M0	M6		Not (M0 or M6)		
1.	LDF	170	0		0		
2.	LDD	170	0		340		
		M0			Not M0		
		Min.	Max.	Typical	Min.	Max.	Typical
3.	ADDF	340	1700	510	680	2040	850
4.	ADDD	340	2890	680	1020	3520	1360
5.	SUBF	340	1700	510	680	2040	850
6.	SUBD	340	2890	680	1020	3570	1360
7.	MULF	850	850	850	1020	1020	1020
8.	MULD	3060	3060	3060	3570	3570	3570
9.	DIVF	6120	6460	—	6290	6800	—
10.	DIVD	11900	12410	—	12240	12580	—
11.	MODF	3040	4250	—	3210	4420	—
12.	MODD	5610	8500	—	6120	9010	—
13.	LDCFD	1870	1870	—	2040	2040	—
14.	LDCDF	2210	2210	—	2720	2720	—
*15.	STF	170	170	—	510	510	—
*16.	STD	170	170	—	510	510	—
*17.	CMPF	170	850	—	340	1020	—
*18.	CMPD	170	850	—	680	1360	—
19.	STCFD	680	850	—	1700	2210	—
20.	STCDF	680	1020	—	1700	2550	—

*For Mode 0 address calculation add 4 cycles

Table B-14 FP11-E Execution Times (Cont.)

Instruction		M0	M6		Not (M0 or M6)		
21.	LDCIF	7310	9860	—	7140	9520	—
22.	LDCID	7310	9690	—	6970	9350	—
23.	LDCLF	7480	10030	—	8500	13770	—
24.	LDCLD	7310	9860	—	8330	13600	—
25.	LDEXP	680	680	—	680	680	—
†26.	STCFI	5270	7650	—	4930	7310	—
†27.	STCFL	5270	10370	—	6800	11900	—
†28.	STCDI	5270	7650	—	4930	7310	—
†29.	STCDL	5270	10370	—	6800	11900	—
30.	STEXP	0	0	—	—	—	—
31.	CLRF	170	170	—	0	0	—
32.	CLRD	170	170	—	0	0	—
33.	NEGF	340	340	—	0	0	—
34.	NEGD	340	340	—	0	0	—
35.	ABSF	340	—	—	0	—	—
36.	ABSD	340	—	—	0	—	—
37.	TSTF	170	—	—	0	—	—
38.	TSTD	170	—	—	0	—	—
39.	LDFPS	0	—	—	0	—	—
40.	STFPS	0	—	—	0	—	—
41.	STST	0	—	—	0	—	—
42.	CFCC	0	—	—	0	—	—
43.	SETF	0	—	—	0	—	—
44.	SETD	0	—	—	0	—	—
45.	SETI	0	—	—	0	—	—
46.	SETL	0	—	—	0	—	—

†Requires CPU shared code execution

LOAD CLASS OF INSTRUCTIONS

CPU		FP11-E
	Load class instruction is fetched. This occurs during previous instruction execution.	
	Instruction is decoded.	
	Address of operands is calculated.	FP11-E decodes instruction and goes into idle state.
	CPU passes operands to the FP11-E.	FP11-E receives operands from CPU.
	Disengage and fetch next instruction.	FP11-E executes instruction.

Figure B-1

LOAD CLASS OF INSTRUCTIONS
(INTEGER CONVERSION)

CPU		FP11-E
	Load class (integer convert) of instructions is fetched. This occurs during previous instruction.	
	Instruction is decoded.	FP11-E decodes instruction, goes into idle state.
	Address of operands is calculated and fetched from memory.	
	Integer conversion by CPU	
	CPU passes result to FP11-E	FP11-E receives result from CPU
	Disengage and fetch next instruction.	FP11-E stores results

Figure B-2

STORE CLASS OF INSTRUCTIONS

CPU		FP11-E
	Store class of instructions is fetched. This occurs during previous instruction.	FP11-E is idle.
	Instruction is decoded.	FP11-E decodes instruction.
	Address of operands is calculated.	FP11-E starts instruction execution.
	CPU waits for FP11-E to complete execution.	
	CPU receives result from the FP11-E and stores it in memory.	FP11-E passes result to be stored in memory.
	CPU fetches next instruction.	FP11-E is idle.

Figure B-3

STORE CLASS OF INSTRUCTIONS
(INTEGER CONVERT)

CPU		FP11-E
	Store class (integer convert) is fetched. This occurs during previous instruction.	FP11-E is idle.
	Instruction is decoded.	FP11-E decodes instruction.
	CPU receives floating point number from FP11-E.	FP11-E passes floating point number.
	Integer conversion performed by CPU.	FP11-E is idle.
	CPU does address calculation and stores result in memory.	

Figure B-4

Figures B-3 and B-4 show, pictorially, how effective execution times for actual floating point instructions in a program are calculated. Note that the effective execution times are dependent on previous floating point instructions. Note also that all memory references are considered to be "cache hits."

Referencing Figure B-5, a sample calculation of effective time would be:
for MULF (R0), AC1

Effective Execution Time is the summation of the following:

Instruction Fetch	340 nsec
Address Calculation Time (Mode 1 from Table B-9)	510 nsec
Wait Time (Since FPP is idle, Wait = 0)	0 nsec
Argument Transfer Time (Transfer 2 words @ 340 nsec/word)	680 nsec
Disengage and Fetch Time	340 nsec
Effective Execution Time for LDF X(R3),AC0 (Ref. Figure B-5)	1870 nsec

First we calculate Wait Time:

Wait Time = [Floating Point Execution (previous FP instruction) (MULF)]	1020 nsec
— [Disengage and Fetch Time (previous FPT instruction)]	— 340 nsec
— [Execution Time of interposing nonFPT instruction (SOB)]	—2400 nsec
— [Instruction Fetch]	— 340 nsec
— [Address Calculation (Mode 6 from Table B-12)]	— 850 nsec
	—2910 nsec

Since calculation resulted in a negative number, Wait Time = 0.

... so Effective Execution Time is the summation of the following:

Instruction Fetch	340 nsec
Address Calculation Time (Mode 6 from Table B-11)	850 nsec
Wait Time (From above calculation)	0 nsec
Argument Transfer Time (2 words @ 340 nsec/word)	680 nsec
Disengage and Fetch Time	340 nsec
Effective Execution Time	2210 nsec

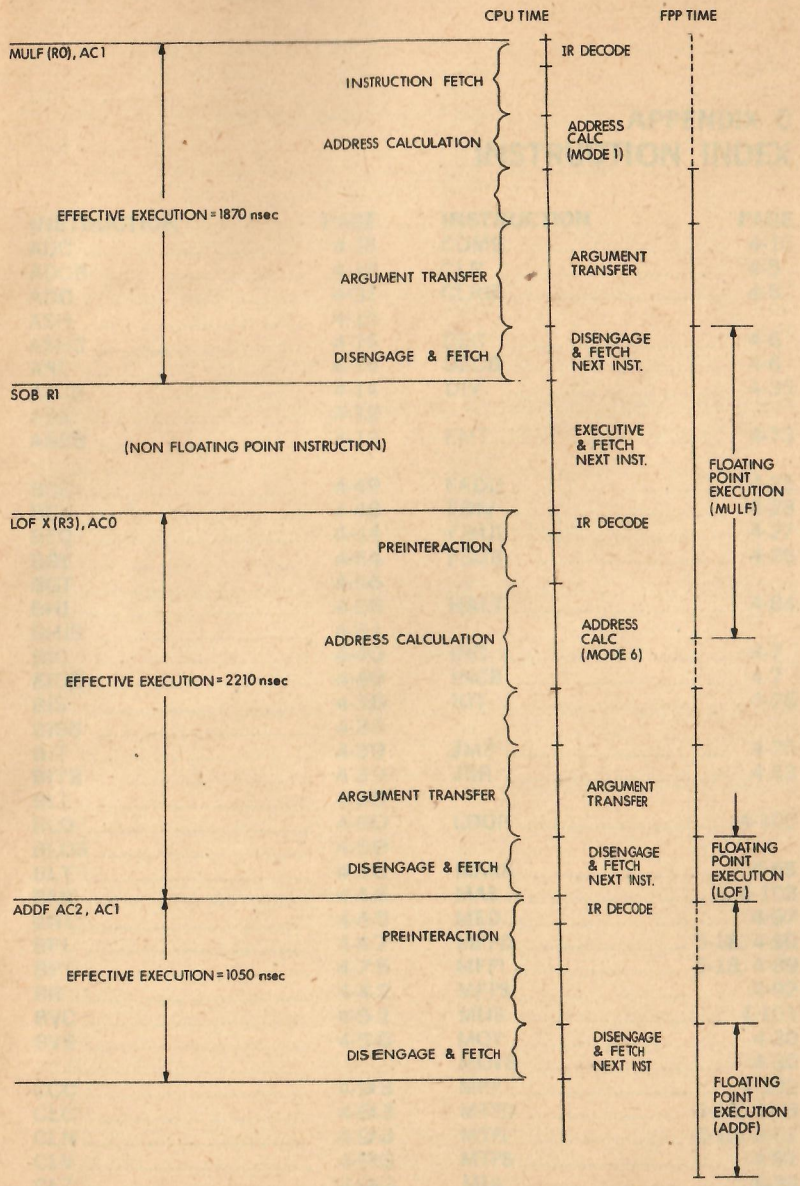


Figure B-5 Calculation of Effective Execution Times for Load Class Instructions

APPENDIX C
INSTRUCTION INDEX

INSTRUCTION	PAGE
ADC	4-18
ADCB	4-18
ADD	4-31
ASH	4-15
ASHC	4-16
ASL	4-14
ASLB	4-14
ASR	4-12
ASRB	4-13
BCC	4-49
BCS	4-48
BEQ	4-44
BGE	4-54
BGT	4-56
BHI	4-58
BHIS	4-61
BIC	4-40
BICB	4-40
BIS	4-38
BISB	4-38
BIT	4-39
BITB	4-39
BLE	4-55
BLO	4-60
BLOS	4-59
BLT	4-53
BMI	4-46
BNE	4-45
BPL	4-47
BPT	4-75
BR	4-42
BVC	4-51
BVS	4-50
CCC	4-93
CLC	4-93
CLN	4-93
CLV	4-93
CLZ	4-93
CMP	4-33
CMPB	4-33
COM	4-10
COMB	4-10
CLR	4-5
CLRB	4-5
DEC	4-6
DECB	4-6
DIV	4-35
EMT	4-73
FADD	4-25
FDIV	4-28
FMUL	4-27
FSUB	4-26
HALT	4-84
INC	4-7
INCB	4-7
IOT	4-76
JMP	4-70
JSR	4-63
LDUP	4-100
MARK	4-65
MAS	4-103
MED	4-97
MFPD	5-18, 4-90
MFPI	5-18, 4-89
MFPS	4-92
MNS	4-101
MOV	4-30
MOVB	4-30
MPP	4-102
MTPD	5-19, 4-88
MTPI	5-19, 4-87
MTPS	4-91
MUL	4-34
NEG	4-8
NEGB	4-8

INSTRUCTION	PAGE
RESET	4-86
ROL	4-22
ROLB	4-22
ROR	4-23
RORB	4-23
RTI	4-77
RTS	4-67
RTT	4-78
SBC	4-19
SBCB	4-19
SCC	4-93
SEC	4-93
SEN	4-93
SEV	4-93
SEZ	4-93
SOB	4-71
SPL	4-69
SUB	4-32
SWAB	4-24
SXT	4-20
TRAP	4-74
TST	4-9
TSTB	4-9
WAIT	4-85
XFC	4-95
XOR	4-36

FLOATING POINT PROCESSOR INSTRUCTION SET

Single Operand Instructions

	PAGE
ABSD	10-36
ABSF	10-36
CFCC	10-41
CLRD	10-35
CLRF	10-35
LDFPS	10-40
NEGD	10-20
NEGF	10-20
SETD	10-38
SETF	10-38
SETI	10-39
SETL	10-39
STFPS	10-40
STST	10-41
TSTD	10-37
TSTF	10-37

Double Operand Instructions

	PAGE
ADDD	10-16
ADDF	10-16
CMPD	10-24
CMPF	10-24
DIVD	10-22
DIVF	10-22
LDCDF	10-28
LDCFD	10-28
LDCID	10-30
LDCIF	10-30
LDCLD	10-30
LDCLF	10-30
LDD	10-14
LDEXP	10-32
LDF	10-14
MODD	10-25
MODF	10-25
MULD	10-21
MULF	10-21
STCDF	10-29
STCDI	10-31
STCDL	10-31
STCFD	10-29
STCFI	10-31
STCFL	10-31
STD	10-15
STEXP	10-34
STF	10-15
SUBD	10-18
SUBF	10-18

NOTES

NOTES

digital

DIGITAL EQUIPMENT CORPORATION, Corporate Headquarters: Maynard, Massachusetts 01754, Telephone: (617)897-5111—SALES AND SERVICE OFFICES: UNITED STATES—ALABAMA, Huntsville • ARIZONA, Phoenix and Tucson • CALIFORNIA, El Segundo, Los Angeles, Oakland, Ridgecrest, San Diego, San Francisco (Mountain View), Santa Ana, Santa Clara, Stanford, Sunnyvale and Woodland Hills • COLORADO, Englewood • CONNECTICUT, Fairfield and Meriden • DISTRICT OF COLUMBIA, Washington (Lanham, MD) • FLORIDA, Ft. Lauderdale and Orlando • GEORGIA, Atlanta • HAWAII, Honolulu • ILLINOIS, Chicago (Rolling Meadows) • INDIANA, Indianapolis • IOWA, Bettendorf • KENTUCKY, Louisville • LOUISIANA, New Orleans (Metairie) • MARYLAND, Odenton • MASSACHUSETTS, Marlborough, Waltham and Westfield • MICHIGAN, Detroit (Farmington Hills) • MINNESOTA, Minneapolis • MISSOURI, Kansas City (Independence) and St. Louis • NEW HAMPSHIRE, Manchester • NEW JERSEY, Cherry Hill, Fairfield, Metuchen and Princeton • NEW MEXICO, Albuquerque • NEW YORK, Albany, Buffalo (Cheektowaga), Long Island (Huntington Station), Manhattan, Rochester and Syracuse • NORTH CAROLINA, Durham/Chapel Hill • OHIO, Cleveland (Euclid), Columbus and Dayton • OKLAHOMA, Tulsa • OREGON, Eugene and Portland • PENNSYLVANIA, Allentown, Philadelphia (Bluebell) and Pittsburgh • SOUTH CAROLINA, Columbia • TENNESSEE, Knoxville and Nashville • TEXAS, Austin, Dallas and Houston • UTAH, Salt Lake City • VIRGINIA, Richmond • WASHINGTON, Bellevue • WISCONSIN, Milwaukee (Brookfield) • INTERNATIONAL—ARGENTINA, Buenos Aires • AUSTRALIA, Adelaide, Brisbane, Canberra, Melbourne, Perth and Sydney • AUSTRIA, Vienna • BELGIUM, Brussels • BOLIVIA, La Paz • BRAZIL, Rio de Janeiro and Sao Paulo • CANADA, Calgary, Edmonton, Halifax, London, Montreal, Ottawa, Toronto, Vancouver and Winnipeg • CHILE, Santiago • DENMARK, Copenhagen • FINLAND, Helsinki • FRANCE, Lyon, Grenoble and Paris • GERMAN FEDERAL REPUBLIC, Cologne, Frankfurt, Hamburg, Hannover, Munich, Nuremburg, Stuttgart and West Berlin • HONG KONG • INDIA, Bombay • INDONESIA, Djakarta • IRELAND, Dublin • ITALY, Milan, Rome and Turin • IRAN, Tehran • JAPAN, Osaka and Tokyo • MALAYSIA, Kuala Lumpur • MEXICO, Mexico City • NETHERLANDS, Utrecht • NEW ZEALAND, Auckland and Christchurch • NORWAY, Oslo • PUERTO RICO, Santurce • SINGAPORE • SPAIN, Madrid • SWEDEN, Gothenburg and Stockholm • SWITZERLAND, Geneva and Zurich • UNITED KINGDOM, Birmingham, Bristol, Epsom, Edinburgh, Leeds, Leicester, London, Manchester and Reading • VENEZUELA, Caracas •

digital

PRINTED IN U.S.A. EB 06498 20/77 050 04 050